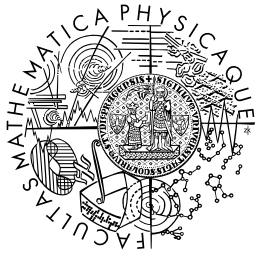


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Monika Švaralová

DRESS & GO: Deep belief networks and Rule Extraction Supported by Simple Genetic Optimization

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	doc. RNDr. Iveta Mrázová, CSc.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2018

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

I would like to thank my supervisor, doc. RNDr. Iveta Mrázová, CSc., for her guidance and the considerable time spent on consultations and proof reading. I would like to thank the faculty for supporting through Student Faculty Grants development of a web application that helped in data collecting and exploration. My thanks also goes to all the volunteers who participated in the data collection, and provided their fashion opinions. Finally, I want to thank Peter for his patience and support during my Master's studies and in writing this thesis.

Title: DRESS & GO: Deep belief networks and Rule Extraction Supported by Simple Genetic Optimization

Author: Bc. Monika Švaralová

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: doc. RNDr. Iveta Mrázová, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Recent developments in social media and web technologies offer new opportunities to access, analyze and process ever-increasing amounts of fashion-related data. In the appealing context of design and fashion, our main goal is to automatically suggest fashionable outfits based on the preferences extracted from real-world data provided either by individual users or gathered from the internet. In our case, the clothing items have the form of 2D-images. Especially for visual data processing tasks, recent models of deep neural networks are known to surpass human performance. This fact inspired us to apply the idea of transfer learning to understand the actual variability in clothing items.

The principle of transfer learning consists in extracting the internal representations formed in large convolutional networks pre-trained on general datasets, e.g., ImageNet, and visualizing its (similarity) structure. Together with transfer learning, clustering algorithms and the image color schemes can be, namely, utilized when searching for related outfit items. Viable means applicable to generating new outfits include deep belief networks and genetic algorithms enhanced by a convolutional network that models the outfit fitness. Although fashion-related recommendations remain highly subjective, the results we have achieved so far prove the viability of this still rather ambitious goal.

Keywords: convolutional neural networks, deep belief networks, genetic algorithms, feature extraction, clustering

Contents

Introduction

Fashion has always been a part of human culture and social structure, as the clothing people have worn often reflected their social status. Fashion is also a large industry, where design and business come together to influence decision-making. In the US alone, the size of the apparel market is $315bn, with women's clothing sales amounting to $110bn annually, according to [Statista, 2017].

The amount of available fashion-related data is growing exponentially due to new technologies, particularly e-commerce and social media. For decades, fashion designs have been documented in images, but nowadays, photos and details of almost every single item of clothing ever produced are being published in e-shops and fashion aggregators. Some of these shopping platforms contain millions of products. Also, on fashion-related blogs and social media sites, people post photos of their outfits, comment and share pictures, together generating vast networks of data.

Like in many fields nowadays, there emerges a trend in the fashion industry towards using methods of data science and artificial intelligence on these vast amounts of data. These methods can be useful for production planning, trend forecasting and targeted clothing recommendation, all with the purpose of optimizing business outcomes. There are multiple AI solutions emerging that target consumers or corporations. There is also a growing amount of research in this area, within companies and in academia, focused especially on the use of image processing and machine learning to achieve these goals. We will explore this further in Chapter 1.4.

Regardless of what one thinks about fashion, every person needs clothing. However, many of the pieces of clothing people buy go unworn for various reasons, so it would be useful to optimize the process of buying clothes to make a more economic choice. Our main goal in this thesis is to propose and implement several strategies of composing wearable outfits from a set of clothing items. This could be either from clothing one already owns, or from products available in a store, which could be helpful in recommending what to buy, based on personal style and the outfits that the person could wear. We will use machine learning methods to learn from the data that is available about clothing and outfits. Composing outfits is a novel topic that is gaining interest as such goals become viable using machine learning. Similar approaches could be used also for other problems of generating combinations of items.

The main information that describes a piece of clothing for us is its image. Recent advances in deep learning on image data have made it easier to extract useful features from images and classify them with a performance on par with humans. In this context, we are going to utilize heavily methods such as convolutional neural networks, training our own networks, and using also large pre-trained models. We will analyze also the color schemes of the images, since color is an important feature of clothing.

In the following Chapters 1 and 2, we will explore preliminaries both in fashion-related area, such as the data that is available, and the theoretical apparatus of machine learning and neural networks that we will be using. In Chapter 3, we will introduce a web application that we created for data collection, browsing and testing. In Chapter 4, we will apply unsupervised clustering methods on color palettes of the images, to extract dominant colors from the image. We will also cluster similar images by color into groups, which can be used when searching for similar outfits.

Chapter 5 is devoted to testing and evaluation of supervised machine learning methods, mainly convolutional neural networks. For image classification in the domain of fashion, we will utilize both large pre-trained models such as Inception v3 [Szegedy et al., 2015] and VGG16 [Simonyan et al., 2014] and small convolutional networks trained from scratch. Using transfer learning from pre-trained models will aid in generalization from the small datasets that we have available. We will explore the representation of clothing items in these networks, and train the models to classify images into fine-grained clothing types. We will also try to predict whether a user would like a clothing item, by classifying a small labeled dataset of clothing rated by the user.

In Chapter 6, we propose and test methods of composing new outfits from a dataset of clothing items using generative models. We designed a general structure of an outfit, which allows us to generate plausible outfits using genetic algorithms and deep belief networks. We introduce a new customized model of joined convolutional networks, which can be used to rate an outfit based on images of the items it contains. We then use a genetic algorithm to arrive at a solution that would be highly rated by such a network. Finally, we test restricted Boltzmann machines and deep belief networks, which are capable of generating new samples from a distribution learned from a dataset of fashionable outfits.

1. Applications of machine learning in fashion e-commerce

1.1 Motivation

Fashion is a way through which people can express their individuality and aesthetics, and also show their association to a certain social group. It is not only a design and cultural phenomenon, but also a large industry. The size of the US apparel market is $315bn, with women's clothing sales amounting to $110bn annually [Statista, 2017]. Yearly spendings per capita on apparel in the U.S. in 2015 were $978.

Shopping for apparel is also a time-consuming activity for many people. A survey of 2,000 women found that they spend each year more than 100 hours shopping for clothes, 40 hours shoe-shopping, and 50 hours window shopping (browsing shops without intending to buy anything) [Johnson, 2015].

Despite the large financial and time costs associated with buying clothes, there is an inefficiency in utilizing the clothing that people own. A study by Marks & Spencer on UK consumers shown that it takes people on average 15 minutes each morning to decide what to wear, with the average closet containing 152 items [Oxfam, 2016]. The study also shown that consumers wear only 44 percent of these items regularly.

1.2 The goal of the thesis

There is a large amount of fashion-related data on the web: in e-shops, catalogs, fashion shows, street photos etc. The goal of this thesis is to analyze and test machine learning methods on such data in order to evaluate the possibility of automating the outfit-composing process. This could also help in providing recommendations what clothes to buy, based on what matches the other items in user's wardrobe. However, it is a complex problem, which requires solving other tasks and leads to questions such as:

- How to structure and represent the highly variable data about fashion products?
- Which fashion products are similar, up to the point of interchangeability? Which ones complement each other?
- What is a fashionable outfit composed of, with regard to the types of clothing, their colors and styles?
- How to accommodate personal style and aesthetics of a person and suggest appropriate clothing or outfits to wear?
- How does the change of fashion trends in time influence these decisions?

- What data can be used to support such a system (product image, text description, clothing types/categories, style attributes, outfits designed by fashion enthusiasts or professional stylists)? How to obtain a large amount of high-quality data from the web?

This thesis will analyze some of these questions and will propose solutions to various necessary subtasks: obtaining the data, finding a good representation and processing them using machine learning algorithms. In the thesis, we will focus on women's fashion, but the same process can be used to analyze men's fashion, in which the products are perhaps less variable.

1.3 Available data

There is an abundance of fashion-related data, especially images, on the web. The data is focused on individual products, or whole outfits, the photos are made in a studio or taken in real-life. Possible sources of a large amount of data may be however unstructured, or corrupted by varying amounts of noise.

Annotated datasets. Academic research made accessible several annotated and cleaned datasets, mostly containing photos of people with bounding areas describing the type of the fashion item in the image, sometimes also the item's attributes.

E-shops, catalogs and shopping platforms. These sources are mostly oriented towards products. Usually, the data have a well categorized structure and manually added descriptions. Each website may have its own structure, although the product categories are similar. The number of items varies, but some of the shopping platforms such as ShopStyle list several million products.

Street style portals. On websites such as Lookbook and Chictopia, users can post photos of themselves in outfits they consider stylish, which can then be rated by other users. This results in a large number of photos, which are sometimes also annotated with colors, tags, or links to the items in e-shops, although these annotations are often noisy and incomplete.

Outfit creators. Sites like Polyvore allow users also to add outfits, but in a schematic way. Such schematic outfits are a collage of products, with links to the catalog photos of the items they contain, usually also along with additional information about each item.

We need to choose and obtain most appropriate datasets for the task. One of our criteria for the dataset selection is that it should not require a lot of manual annotation and cleaning. It should be possible to automatically obtain the dataset by crawling, or be available openly for download. Further, it should be also large enough, to allow us to use only a small fraction of it for testing purposes, and to be able to scale it up later on to obtain improved results with more advanced machine learning algorithms.

1.3.1 Polyvore

Polyvore [Polyvore, 2017] is a clothing aggregator with a large database of clothing items, but it also allows users to compose outfits (or collages which may not represent actual outfits) from these items. There are many fashion enthusiasts, as well as expert stylists, creating high-quality outfits. Website with an outfit contains links to catalog photos of items that the outfit contains, which usually provide also additional information about the item, such as its text description, category, brand and price.

An example of such an outfit, along with the collection of items that it contains, is shown in Figure 1.1. There may occur errors in descriptions, and sometimes also in the content of an outfit, because the collage of items created by user does not have to be a complete, wearable outfit. These outfits and clothing items are also voted on by other users, which can yield a measure (although not always a reliable one) of their stylishness.

The categories and subcategories of women's clothing, shoes and accessories at Polyvore is shown in Table 1.1. The information about the categories and subcategories of each item are of advantage to us, since they determine the role of the items in an outfit, as well as a person's style or occasion to wear the outfit.

In this thesis, we have used [Polyvore, 2017] as our main source of data, and many of the figures in the text will be adapting images from this scraped collection of data.

1.4 Related work

Machine learning and computer vision research related to fashion has in the past few years started receiving increasing interest due to its real-life applications, an enormously large market associated with fashion and increasingly large amounts of data available.

This section provides an overview of related work and previous research related to fashion e-commerce, clothing, and similar design-oriented topics. Following are discussed the topics most relevant to (sub-)problems of the thesis, mostly regarding data from fashion industry.

1.4.1 Fashion item classification, attribute recognition or segmentation

A common topic in published research related to machine learning in fashion includes classification of fashion images, or recognizing certain attributes based on a manually labeled dataset of images [Bossard et al., 2013; Hara et al., 2016; Liu et al., 2016]. In some cases, also segmentation, or finding the position of the fashion item in an image, is applied.

(a) **Leo Print** #leopard-print #animalprint #brown #mimicdesign

(b) **Tibi - Merino Wool Cropped Sweater** The soft-on-skin Merino Wool Structured Sweater from Tibi is completely cozy and 100% fashion forward! Styled with cropped length, ribbed edges and a crisp fall-ready amber hue.

(c) **Boyfriend Low Ripped Jeans $39.99** Denim blue. 5-pocket, low-rise jeans in washed denim with heavily distressed details, button fly, and slightly wider, tapered legs.

(d) **Aquazzura Christy 75 Suede Pump** Combining timeless elegance with contemporary style, Aquazzura's Christy 75 pump is crafted from rich suede with a sleek pointed toe, ghillie lace-up design and accentuated ankle. Resting on a comfortable mid-height heel, it will inject Italian finesse into day and evening ensembles.

(e) **Leopard fold over clutch, Leopard Clutch, Leather leopard clutch, Lepo...** Leopard-print calf-hair puts a modern twist on this signature fold over clutch! The brass metal zip opens to a lined interior! PRODUCT INFO size:

(f) **Michael Kors Kacie Gold Sunray Dial Ladies Watch Set MK3568** Shop for Kacie Gold Sunray Dial Ladies Watch Set by Michael Kors at JOMASHOP for only $193.13! WARRANTY or GUARANTEE available with every item. We are the internet's leading source for ! (Model # MK3568)

Figure 1.1: An example of an outfit from the Polyvore dataset, and the items it contains, along with the title and description for each of them. Item descriptions usually come from an e-shop or can be user-provided. They do not follow the same structure. They can contain information about style, color, material, or brand, but often also irrelevant words.

Women's Fashion	Women's Shoes	Women's Accessories
• Dresses – Day – Cocktail – Gowns • Skirts – Mini – Knee-length – Long • Tops – Blouses – Cardigans – Sweaters – Sweatshirts and hoodies – Tanks – T-shirts – Tunics ...	• Athletic • Boots – Ankle bootie – Mid calf – Knee high – Over-the-knee • Clogs • Flats • Loafers and Moccasins • Oxfords • Pumps • Sandals • Sneakers	• Bags – Backpacks – Handbags – Messenger bags – Wallets • Jewelry – Bracelets and bangles – Brooches – Charms and pendants – Earrings – Necklaces – Rings – Watches ...

Table 1.1: Categories and subcategories of women's fashion products on Polyvore

In [Liu et al., 2016], a dataset of 800,000 images is introduced, with type and attribute annotations and image landmarks, where a model based on convolutional networks learns clothing features by jointly predicting clothing attributes and landmarks. Models based on convolutional networks are often successful, along with pose detection of the person in the image [Hara et al., 2016], because certain items have higher probability to be at a specific part of the pose, e.g., bag near hand.

1.4.2 Recognizing style, matching products

In addition to recognizing attributes of clothing, style annotation can be generated. [Yamaguchi et al., 2015] studies also the compatibility of clothing items and attributes. [Liu et al., 2012] suggests clothing recommendations based on occasion and aesthetics. In [Di et al., 2013], fine-grained attributes for clothing retrieval based on style are trained. [Simo-Serra et al., 2015] models the perception of fashionability in photos, providing suggestions to a user what changes to make in order to improve fashionability.

1.4.3 Visual search, feature representation

A highly coveted task in this area is recognition of fashion items from photos in a natural environment. This has a real-life application, because when people see a photo with a clothing item they like, they become interested in buying that item. Without the ability to search by image, it becomes difficult to find the exact same item.

In [Kalantidis et al., 2013] and [Kiapour et al., 2015], clothing detection in an image and retrieval of the detected item from a dataset of products is implemented. Pinterest implemented a similar visual retrieval of products found in photos in [Jing et al., 2015]. [Bell et al., 2015] focused of finding similar items of home design, training a Siamese convolutional network to embed the same item into representations close to one another. In [Zoghbi et al., 2016], cross-modal (image and text) search and retrieval is implemented.

1.4.4 Personalized fashion product recommendation

[Bracher et al., 2016] uses sales data for fashion recommendation to customers of Zalando shopping platform. It is possible to take clothing recommendation one step further, and to generate new fashion items based on knowledge of customers' preferences. In [Kang et al., 2017], Siamese CNNs and generative adversarial networks (GANs) have been applied for generating new images that are most consistent with a user's personal taste.

1.4.5 Outfit composition

Only around 2017, the task of composing outfits started to be explored in research papers, often using data from Polyvore sets. [Li et al., 2016] is the first example where a goal of outfit composition, similar to the one we have in this thesis, is proposed. A dataset of Polyvore sets is used, with image data and metadata of each clothing item, and users' rating as the target measure. Multimodal embedded representations of each item are used, with a multilayer perceptron predicting a set's popularity. [Han et al., 2017] focuses also on generating outfits, training a LSTM model to predict the next item a sequence of items in an outfit.

[Tangseng et al., 2017] shows an approach to generating outfits based on items in a person's closet, using a large dataset of Polyvore outfits and extracting representations from a pre-trained convolutional network. In [Hsiao et al., 2017], an approach to generating capsule wardrobes, that is, a minimal set of clothing items with which a large number of outfits can be composed, has been proposed. [Song et al., 2017] proposed an approach to matching tops and bottoms using outfit data from Polyvore.

2. Preliminaries

2.1 Clustering methods

The goal of clustering algorithms is to partition data points into several groups called clusters in such a way that the observations assigned to the same cluster are more similar to one another than those assigned to different clusters [Hastie et al., 2009]. Each of the N observations is uniquely labeled by an integer $i \in 1, ..., N$. A number of clusters $K < N$ is specified, and each cluster is labeled by an integer $k \in 1, ..., K$. The assignments of observations into clusters can be characterized by an encoder $C(i) = k$, which assigns the ith observation to the kth cluster.

To evaluate the cluster assignments, we can specify a loss function W, which we attempt to minimize through the use of a clustering algorithm:

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} d(\mathbf{x}_i, \mathbf{x}_{i'}) \tag{2.1}$$

This function is sometimes referred to as within-cluster point scatter. It measures the dissimilarity of points within each cluster, or how distant the points within each cluster are to one another. The vectors $\mathbf{x}_i$ and $\mathbf{x}_{i'}$ represent each pair of points assigned to the same cluster. $d(\mathbf{x}_i, \mathbf{x}_{i'})$ is a dissimilarity measure; Euclidean distance is often used.

2.1.1 K-means

K-means is a simple and commonly used clustering algorithm. Its objective is to minimize the loss function, using Euclidean distance as the dissimilarity measure:

$$W(C) = \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} \|\mathbf{x}_i - \mathbf{x}_{i'}\|^2 \tag{2.2}$$

$$= \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|\mathbf{x}_i - \bar{\mathbf{x}}_k\|^2 \tag{2.3}$$

where $\bar{\mathbf{x}}_k$ is the representative vector associated with the kth cluster, calculated as a centroid of the points assigned to the cluster. N_k is the number of observations assigned to the kth cluster.

The standard k-means algorithm takes an iterative approach. First, the K cluster representatives $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_K$ are initialized. There are a few methods for initialization, one of them is to select K random observations from the data, and use them as the initial representatives.

The following steps are performed until cluster assignments do not change:

1. Given a current set of representatives $\{\mathbf{m}_1, ..., \mathbf{m}_K\}$, we assign each observation $\mathbf{x}_i$ to the closest representative:

$$C(i) = \underset{k}{\operatorname{argmin}} \|\mathbf{x}_i - \mathbf{m}_k\| \tag{2.4}$$

2. Calculate the new cluster representatives $\{\mathbf{m}_1, ..., \mathbf{m}_K\}$ as centroids of the observations in each of the clusters:

$$\mathbf{m}_k = \frac{1}{N_k} \sum_{C(i)=k} \mathbf{x}_i \tag{2.5}$$

The algorithm iteratively minimizes the loss function (2.2) and converges to a local optimum. Because the resulting clusters depend on the initial choice of representatives, it should be initialized multiple times from randomly chosen cluster centers, and return the solution which has the smallest value of the objective function.

2.1.2 Hierarchical clustering

A disadvantage of k-means algorithm is that we have to specify the number of clusters in advance. Hierarchical methods do not need this parameter, but they usually require a specification of some target measure of dissimilarity between clusters.

There are two basic paradigms of hierarchical clustering: agglomerative (bottom-up) and divisive (top-down). The results of these methods can be visualized in the form of a dendrogram, which is a tree-like diagram where the lengths of branches correspond to the degree of dissimilarity between nodes.

Agglomerative clustering

Agglomerative clustering starts with every observation representing a single cluster. At each steps, two closest clusters (according to the chosen dissimilarity metric) are merged into a single cluster.

Let G and H represent two such groups of observations. The dissimilarity $d(G, H)$ is computed from the set of pairwise observation dissimilarities $d_{ii'}$ where $i \in G$ and $i' \in H$. The most commonly used dissimilarity metrics are:

Single linkage The dissimilarity of two clusters is that of the closest pair of observations in the two clusters:

$$d_{SL}(G, H) = \min_{i \in G, i' \in H} d_{ii'} \tag{2.6}$$

where $d_{ii'} = d(\mathbf{x}_i, \mathbf{x}_{i'})$.

Complete linkage takes the dissimilarity of the furthest pair:

$$d_{CL}(G, H) = \max_{i \in G, i' \in H} d_{ii'} \tag{2.7}$$

Group average uses the average dissimilarity of observations between clusters:

$$d_{GA}(G, H) = \frac{1}{N_G N_H} \sum_{i \in G} \sum_{i' \in H} d_{ii'} \tag{2.8}$$

Clustering can be produced by terminating the procedure when the dissimilarity of groups that are to be merged exceeds a threshold value.

2.2 Neural networks

A *neural network* is a multilayer stack of modules, which are subject to learning [LeCun et al., 2015]. Each module consists of several units or neurons, each of which computes a non-linear mapping from input to output. Trainable parameters of the network are called weights.

The training sample, which is used to train a network in a supervised manner, consists of N pairs of input and target output [Haykin, 2008]:

$$\mathscr{T} = \{\mathbf{x}(n), \mathbf{d}(n)\}_{n=1}^{N} \tag{2.9}$$

In multilayer neural networks, training is usually performed using gradient descent method. In such a case, we define an objective function $\mathscr{E}(n)$ which measures the error or distance between the actual outputs $y_j(n)$ of the network when input stimulus $\mathbf{x}(n)$ is applied to the input layer and the desired outputs $d_j(n)$. We then attempt to minimize this function:

$$\mathscr{E}(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n) \tag{2.10}$$

where

$$e_j(n) = d_j(n) - y_j(n) \tag{2.11}$$

represents the error signal at the output of neuron j. Set C includes all neurons in the output layer.

To adjust the weights of the network, the learning algorithm computes a gradient vector that represents the change in the error function with respect to the change of each weight. The weight vector is then adjusted in the direction opposite to the gradient.

There are two different modes of training – the so-called batch and online (stochastic) gradient descent. Stochastic gradient descent (SGD) computes the outputs and errors for one example from the training set, computes the gradient for this example and adjusts the weights. This is repeated until the error stops decreasing. In batch learning, adjustments are performed after all N training samples have been presented, averaging their resulting gradients. In practice, mini-batches, which are small subsets of the training sample, are often used. Each set of examples gives an estimate of the average gradient for the whole training set.

2.2.1 Backpropagation

Several neural network models, such as the multilayer perceptron, use feedforward architecture, which maps a fixed-size input to a fixed-size output (Fig. 2.1). Each unit in a layer computes a weighted sum of its inputs from the previous layer, and passes it through a non-linear transfer function. Units that are not in the input or output layer are called hidden units.

The inner potential $\upsilon_j(n)$ of neuron j when input $\mathbf{x}(n)$ is applied to the input layer is

$$\upsilon_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n) \tag{2.12}$$

where m is the total number of inputs to neuron j. $y_i(n)$ is the output of the neuron i in the previous layer, which is connected with neuron j through a synaptic weight w_{ji}. The synaptic weight w_{j0} represents the bias b_j corresponding to a fixed input $y_0 = +1$.

The output of neuron j is

$$y_j(n) = \varphi(\upsilon_j(n)) \tag{2.13}$$

where $\varphi(z)$ is the non-linear transfer function. Some of the commonly used functions are hyperbolic tangent $tanh(z)$, log-sigmoid $logsig(z) = 1/(1 + e^{-z})$ and rectified linear unit (ReLU) $f(z) = max(z, 0)$. These functions are smooth, non-decreasing and differentiable.

Backpropagation algorithm computes the gradient of the objective function with respect to the weights of the network. It is an application of the chain rule for derivatives. The derivative of the objective function with respect to the input of a neuron can be computed from the gradient with respect to the output of that neuron. It can be applied repeatedly to propagate the error terms from the output layer backwards to the input layer.

We can express the gradient of the objective function $\mathscr{E}$ with respect to the weight w_{ji} as

$$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathscr{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial \upsilon_j(n)} \frac{\partial \upsilon_j(n)}{\partial w_{ji}(n)} \tag{2.14}$$

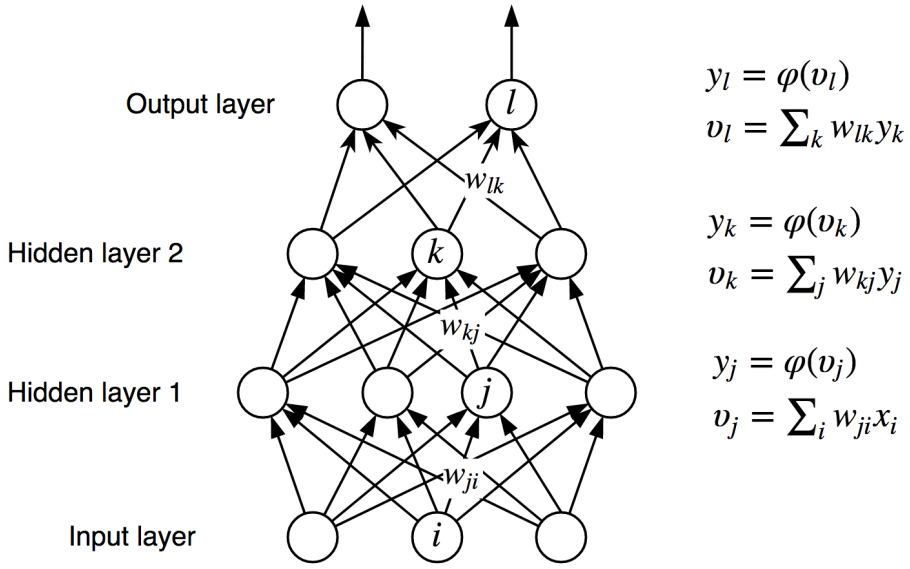


Figure 2.1: A neural network with two hidden layers and the equations used to compute the output. At each layer we compute the inner potential v of each unit, and then we apply a nonlinear function $\varphi(v)$ to get the output of the unit. Bias terms have been omitted for simplicity.

Differentiating Eq. 2.10 with respect to $e_j(n)$, we obtain

$$\frac{\partial \mathscr{E}(n)}{\partial e_j(n)} = e_j(n) \tag{2.15}$$

Differentiating Eq. 2.11 with respect to $y_j(n)$, we obtain

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{2.16}$$

Differentiating Eq. 2.13 with respect to $v_j(n)$, we obtain

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n)) \tag{2.17}$$

Finally, differentiating Eq. 2.12 with respect to w_{ji}, we obtain

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \tag{2.18}$$

Substituting these equations into Eq. 2.14 yields

$$\frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n) \tag{2.19}$$

The local gradient $\delta_j(n)$ is defined by:

$$\begin{aligned}\delta_j(n) &= \frac{\partial \mathscr{E}(n)}{\partial \upsilon_j(n)} \\ &= e_j(n)\varphi_j'(\upsilon_j(n))\end{aligned} \tag{2.20}$$

This formula for the local gradient can be used when neuron j is in the output layer, because we can directly compute e_j using Eq. 2.11. For neuron j in the hidden layer, with a sequence of differentiations we can obtain the backpropagation formula for the local gradient:

$$\delta_j(n) = \varphi_j'(\upsilon_j(n)) \sum_k \delta_k(n) w_{kj}(n) \tag{2.21}$$

The correction Δw_{ji} applied to the weight w_{ji} is

$$\begin{aligned}\Delta w_{ji}(n) &= -\eta \frac{\partial \mathscr{E}(n)}{\partial w_{ji}(n)} \\ &= \eta \delta_j(n) y_i(n)\end{aligned} \tag{2.22}$$

where η is the learning-rate parameter of the algorithm. Using the minus sign we adjust the weight in the opposite direction of the gradient, seeking the minimum of the objective function.

The algorithm cannot be shown to converge. Usually the training is stopped when some predefined criteria is fulfilled, e.g., the rate of change in the error is very small. There are various methods of optimizing the original gradient descent algorithm, which use adaptive learning rates and are more computationally efficient, such as AdaGrad [Duchi et al., 2010] and Adam [Kingma et al., 2014].

2.3 Convolutional networks

Convolutional neural networks (CNN) are a specialized kind of neural network for processing data that has a grid-like topology. An example of these represent image data, which can be thought of as a 2D grid of pixels [Goodfellow et al., 2016]. CNNs are neural networks that use a mathematical operation called convolution instead of a general matrix multiplication in at least one of their layers. They have been very successful in practical applications.

2.3.1 Convolution

Convolution is an operation on two functions with real-valued arguments. For example, we have noisy measurements $x(t)$ of the position of an object in time t, and a weighting function $w(a)$, where a represents the age of the measurement. If we apply a weighted measurement at each moment, we can obtain a new function s providing a smoothed estimate of x in time t:

$$s(t) = \int x(a)w(t-a)da \tag{2.23}$$

$$= (x * w)(t) \tag{2.24}$$

This operation is called convolution. In convolutional networks, the first argument (function x) is often referred to as the input and the second argument (function w) as the kernel.

Usually, when working with real-life data, time will be dicretized and the index t will take on only integer values. We can define the discrete convolution as follows:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.25}$$

In practice, we replace the infinite summation by a finite number of array elements. Usually in machine learning applications, the input is a multidimensional array of data and the kernel is a multidimensional array of parameters that are adapted by the learning algorithm. These arrays are referred to as tensors. We can use convolutions over more than one axis at a time. For example, with two-dimensional image I as an input, and two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.26}$$

Convolution is a commutative operation, therefore

$$(I * K) = (K * I) \tag{2.27}$$

Convolution leverages three ideas that can help improve a machine learning system:

Sparse interactions Because the kernel is typically smaller than the input, we need to store fewer parameters, which reduces the memory requirements and improves statistical efficiency of the model. Computing the output also requires fewer operations, leading to large improvements in efficiency.

Parameter sharing refers to using the same set of parameters for more than one function in a model. This further reduces memory requirements of the model. Each member of the kernel is used at every position of the input (except some of the boundary pixels).

Equivariant representations Convolutional layers have a property called equivariance to translation. On the other hand, convolutions are not naturally equivariant to some other transformations, such as scaling or rotation.

For example with images, convolution creates a 2D map of where certain features appear in the input. It is useful to detect edges in the first layer of a convolutional network. Since we usually want to recognize the same edges everywhere in the image, it is practical to share the parameters that recognize edges across the entire image.

2.3.2 Pooling

In a convolutional network, a layer typically works in three stages:

1. Several convolutions are performed in parallel, which produces a set of linear activations.

2. Each linear activation is run through a nonlinear activation function (e.g the rectified linear activation function). This is called a detector stage.

3. We use a pooling function to further modify the output of the previous layer.

The pooling function calculates a summary statistic of nearby outputs at a certain location of the network. For example, max pooling operation returns the maximum output within a rectangular neighborhood. Pooling helps to make the representation approximately invariant to small translations of the input. This property can be very useful if we care more about whether some feature is present than exactly where it is.

Because pooling summarizes the outputs over a whole neighborhood, it is possible to use fewer pooling units than detector units. In this way, we can summarize the statistics for regions spaced several pixels apart. This also improves the computational efficiency.

2.3.3 Architecture and properties of CNNs

The architecture of a CNN usually consists of many convolutional and pooling layers. If we use the convolutional network for classification, then at the final stage the output tensor is reshaped so its spatial dimensions flatten out. This is provided as an input to a feedforward network classifier. One example of using CNN for classification is illustrated in Fig. 2.2.

Usually, the convolutional layer performs multiple convolutions in parallel, because we want the layer to extract many kinds of features, at many locations. Convolution with a single kernel can only extract one kind of feature.

The input is not just a grid of real values, but of vectors. For example, each pixel of a color image has a certain intensity of red, green and blue channel. In the further layers, the input to a layer is the output from many different convolutions of the previous layer. In the case of images, we can think of them to be represented as 3D tensors.

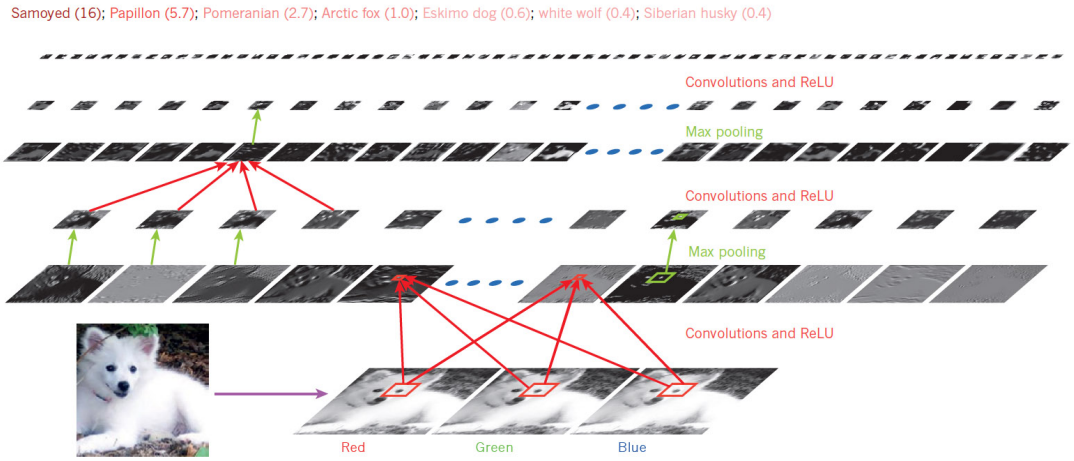


Figure 2.2: The outputs of each layer of a typical convolutional network, applied to an image of a Samoyed dog. Each output image is a feature map corresponding to one of the learned features, where lower-level features act mainly as oriented edge detectors. Final output at the top represents the score for each of the considered classes. Reproduced with permission from [LeCun et al., 2015].

Let us assume that we have an input image represented as a tensor $\mathbf{V}$, where $V_{i,j,k}$ represents the input value at color channel i, and pixel at the row j and column k. The output (image) $\mathbf{Z}$ has the same format as $\mathbf{V}$. We need a 4D kernel tensor $\mathbf{K}$ with each element $K_{i,j,k,l}$ representing the connection strength between the unit in the channel i of the output and the channel j of the input, with an offset of k rows and l columns between the output and the input unit. We can compute the output $\mathbf{Z}$ as

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n} \tag{2.28}$$

where the summation is over all values for which the tensor indexing is valid.

We can skip over some positions of the kernel in order to reduce the computational cost, and calculate the output at positions spaced s pixels in each direction, referring to s as the stride. We can define a downsampled convolution function c:

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \tag{2.29}$$

We generally also add a bias term to each output before applying the nonlinear function. It is typical to have one bias per channel of the output and share it across all location. It is also possible to learn a separate bias at each location.

2.3.4 Training a convolutional network

Let us suppose we want to train a CNN that incorporates strided convolution defined by $c(\mathbf{K}, \mathbf{V}, s)$, by means of minimizing some loss function $J(\mathbf{V}, \mathbf{K})$. During forward propagation, we compute the output $\mathbf{Z}$ using the convolution function

c, which is then used to compute the cost function J. During back-propagation, we will receive a tensor $\mathbf{G}$ such that

$$G_{i,j,k} = \frac{\partial J(\mathbf{V}, \mathbf{K})}{\partial Z_{i,j,k}} \tag{2.30}$$

To train the network, we need to compute the gradient with respect to the weights in the kernel, given the gradient with respect to the outputs:

$$g(\mathbf{G}, \mathbf{V}, s)_{i,j,k,l} = \frac{\partial J(\mathbf{V}, \mathbf{K})}{\partial K_{i,j,k,l}} = \sum_{m,n} G_{i,m,n} V_{j,(m-1)\times s+k,(n-1)\times s+l} \tag{2.31}$$

If this layer is not the bottom one of the network, we need to compute the gradient with respect to $\mathbf{V}$ in order to back-propagate the error further down. We can use the function

$$h(\mathbf{K}, \mathbf{G}, s)_{i,j,k} = \frac{\partial J(\mathbf{V}, \mathbf{K})}{\partial V_{i,j,k}} \tag{2.32}$$

$$= \sum_{\substack{l,m \\ \text{such that} \\ (l-1)\times s+m=j}} \sum_{\substack{n,p \\ \text{such that} \\ (n-1)\times s+p=k}} \sum_{q} K_{q,i,m,p} G_{q,l,n} \tag{2.33}$$

One advantage of CNNs is that they can process inputs of various sizes, e.g., images with different width and height. The kernel is then applied a different number of times depending on the size of the input.

CNNs can be also used to output a high-dimensional, structured object, instead of just predicting a class label for a classification task. For example, the model might output a tensor $\mathbf{S}$, where $S_{i,j,k}$ is the probability that pixel (j, k) of the input belongs to class i.

2.4 Generative models

2.4.1 Restricted Boltzmann machines

Restricted Boltzmann machines (RBM) introduced in [Smolensky, 1986] are undirected probabilistic graphical models containing a layer of observable variables and a single layer of latent variables. It is a bipartite graph, where no connections are permitted within a layer (Figure 2.3).

In the case of a binary RBM, the observable layer consists of a vector $\mathbf{v}$ of n_v binary random variables, and the hidden layer of vector $\mathbf{h}$ of n_h binary random variables. Its joint probability distribution is specified by the energy function [Goodfellow et al., 2016]:

$$P(\mathbf{v} = \boldsymbol{v}, \mathbf{h} = \boldsymbol{h}) = \frac{1}{Z} \exp(-E(\boldsymbol{v}, \boldsymbol{h})) \tag{2.34}$$

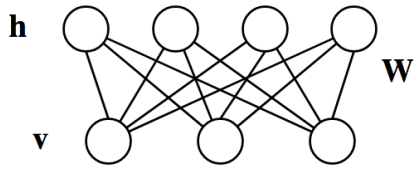


Figure 2.3: Schematic diagram of a restricted Boltzmann machine

The energy function E is given by

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{b}^\top \boldsymbol{v} - \boldsymbol{c}^\top \boldsymbol{h} - \boldsymbol{v}^\top \boldsymbol{W} \boldsymbol{h} \tag{2.35}$$

and Z is the normalizing constant

$$Z = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} \exp -E(\boldsymbol{v}, \boldsymbol{h}) \tag{2.36}$$

RBMs are trained to maximize the product of probabilities of some training set $\boldsymbol{V}$, where each row corresponds to a visible vector $\boldsymbol{v}$:

$$\underset{\boldsymbol{W}}{\operatorname{argmax}} \prod_{\boldsymbol{v} \in \boldsymbol{V}} P(\boldsymbol{v}) \tag{2.37}$$

The algorithm most often used to compute the weight vector $\boldsymbol{W}$ is contrastive divergence. It performs Gibbs sampling [Geman et al., 1984] and is used inside a gradient descent procedure to compute weight update.

2.4.2 Deep belief networks

Deep belief networks (DBN) [Hinton et al., 2006] are generative models based on restricted Boltzmann machines. A DBN consists of several layers of hidden units, which typically take on binary values, and a layer of visible units, which can have binary or real values. Every two neighboring layers are fully connected and there are no connections between units from the same layer. The connections between the top two layers are undirected, whereas all the other connections are directed [Goodfellow et al., 2016]. A schematic diagram of a DBN is shown in Figure 2.4.

A DBN with l hidden layers contains l weight matrices $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(l)}$ and $l+1$ bias vectors $\mathbf{b}^{(0)}, \ldots, \mathbf{b}^{(l)}$, where $\mathbf{b}^{(0)}$ provides the biases for the visible layer. The probability distribution represented by the DBN is

$$P(\mathbf{h}^{(l)}, \mathbf{h}^{(l-1)}) \propto \exp\left(\mathbf{b}^{(l)\top} \mathbf{h}^{(l)} + \mathbf{b}^{(l-1)\top} \mathbf{h}^{(l-1)} + \mathbf{h}^{(l-1)\top} \mathbf{W}^{(l)} \mathbf{h}^{(l)}\right) \tag{2.38}$$

for the topmost pair of layers with undirected connections, and

$$P(h_i^{(k)} = 1 \mid \mathbf{h}^{(k+1)}) = \sigma\left(b_i^{(k)} + \mathbf{W}_{:,i}^{(k+1)\top} \mathbf{h}^{(k+1)}\right) \forall i, \forall k \in 1, \ldots, l-2 \tag{2.39}$$

$$P(v_i = 1 \mid \mathbf{h}^{(1)}) = \sigma\left(b_i^{(0)} + \mathbf{W}_{:,i}^{(1)\top} \mathbf{h}^{(1)}\right) \forall i \tag{2.40}$$

for all the remaining pairs of layers, which have directed connections.

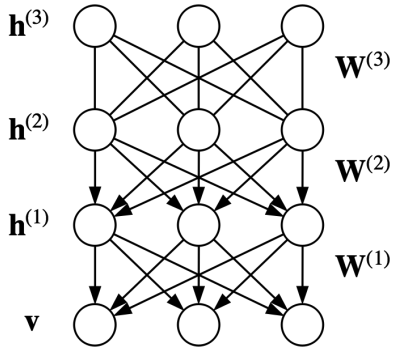


Figure 2.4: An example of a deep belief network with three hidden layers $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}$ and a visible layer $\mathbf{v}$ at the bottom, each consisting of three units. Each pair of neighboring layers $\mathbf{h}^{(k-1)}$ and $\mathbf{h}^{(k)}$ is mutually interconnected by means of weights forming the weight matrix $\mathbf{W}^{(k)}$. Only the topmost pair of layers has undirected connections.

In order to generate a sample from a DBN, we first need to run several steps of Gibbs sampling on the top two layers. Then, we use a single pass of sampling through the rest of the model to draw a sample from the visible units.

To train a DBN, we begin by training an RBM to maximize $\mathbf{E}_{\mathbf{v} \sim p_{data}} \log p(\mathbf{v})$ using contrastive divergence or stochastic maximum likelihood. The resulting parameters then define the first layer of the DBN. Next, we train a second RBM to approximately maximize

$$\mathbf{E}_{\mathbf{v} \sim p_{data}} \mathbf{E}_{\mathbf{h}^{(1)} \sim p^{(1)}(\mathbf{h}^{(1)} \mid \mathbf{v})} \log p^{(2)}(\mathbf{h}^{(1)}) \tag{2.41}$$

where $p^{(1)}$ and $p^{(2)}$ are the probability distributions represented by the first and the second RBM, respectively. While the first RBM is driven by the data, the second is trained to model the distribution defined by sampling the hidden units of the first RBM. This process can be repeated to add multiple layers of RBMs to the DBN.

The trained DBN can either be used directly as a generative model, or it can be used to improve classification models. We can define a multilayer perceptron using the trained weights and biases from the DBN. The output of the neurons from layer $\mathbf{h}^{(l)}$ can be determined according to:

$$\mathbf{h}^{(1)} = \sigma\left(b^{(1)} + \mathbf{v}^\top \mathbf{W}^{(1)}\right) \tag{2.42}$$

$$\mathbf{h}^{(k)} = \sigma\left(b^{(k)} + \mathbf{h}^{(k-1)\top} \mathbf{W}^{(k)}\right) \forall k \in 2, \ldots, l \tag{2.43}$$

After initializing a multilayer perceptron with the weights and biases from the corresponding DBN, we can further train it, e.g. by the backpropagation algorithm, to perform a classification task.

2.5 Genetic algorithms

Genetic algorithms (GAs) provide a learning method based loosely on biological evolution [Mitchell, 1997]. It is an optimization method that searches a large space of candidate hypotheses seeking one that has the best performance according to a predefined fitness function. GA performs a randomized, parallel search in the hypothesis space.

There are various different implementations of GAs, which typically share the following structure: the algorithm iteratively updates a pool of hypotheses, called the population. In each iteration, a fitness function is evaluated for each member of the population. A new population is generated by probabilistically selecting the individuals with the highest fitness from the current population. Some of these individuals are carried over to the next population, others are used as a basis for creating new offspring individuals using genetic operations such as crossover and mutation. A prototype of a GA is described in Algorithm 1.

Algorithm 1 A genetic algorithm prototype [Mitchell, 1997]

function GA($Fitness, Fitness\_threshold, p, r, m$)
$Fitness$: A function that assigns an evaluation score to a hypothesis
$Fitness\_threshold$: A threshold specifying the termination criterion
p: The number of hypotheses (individuals) in the population
r: Crossover rate
m: Mutation rate

Initialize population: $P \leftarrow$ Generate p hypotheses at random
Evaluate: For each h in P, compute $Fitness(h)$
while $[max_h Fitness(h)] < Fitness\_threshold$ **do**
Select: Probabilistically select $(1-r)p$ members of P to add to the new generation, P_s
Crossover: Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P. For each pair (h_1, h_2), produce two offspring by applying the Crossover operator, and add the offspring to P_s.
Mutate: Choose random m percent of P_s and apply Mutation operator to each of the selected hypotheses
Update: $P \leftarrow P_s$
Evaluate: for each h in P, compute $Fitness(h)$
end while
return the hypothesis from P which has the highest fitness
end function

Hypotheses in GAs are often represented by bit strings. Various attributes can be encoded into bit strings, which can be manipulated by genetic operators. In some GAs, hypotheses are represented by symbolic description. For example, in genetic programming, hypotheses are encoded as computer programs.

2.5.1 Genetic operators

The generation of successors in a GA is determined by operators which recombine and mutate selected members of the current population. The two most common operators are *crossover* and *mutation*.

The crossover operator produces two new offspring from two selected parent strings, by combining bits from each parent. The most common ways to combine bit strings are by randomly choosing the point(s) of a crossover: single-point crossover, two-point crossover and uniform crossover.

The mutation operator produces small random changes to a bit string by choosing a position at random at which it changes the value.

2.5.2 Fitness function and selection

The fitness function defines a criterion for ranking potential hypotheses and probabilistically selecting them to the next generation. The probability of selection of an individual can be defined in several ways:

Roulette wheel selection – probability of hypothesis selection is proportional to the ratio of its fitness to the fitness of other members of the population.

Tournament selection – two hypotheses are chosen at random from the current population. With predefined probability p, the more fit hypothesis is selected, and with probability $(1 - p)$, the less fit hypothesis is selected.

Rank selection – hypotheses in the population are sorted by fitness, the probability of hypothesis selection is proportional to the rank in this list instead of its fitness.

3. Web Application and Data Collection

The need to visually explore the dataset and test recommendation models, as well as to obtain new, subjective data from volunteers, led us to developing a prototype of a web application that would contain all of these features. This prototype of an application can be further enriched with machine learning algorithms and used for their testing. It was implemented in Django [Django Software Foundation, 2017], an MVC (Model-view-controller) framework in Python.

The main purposes of the application are:

- Obtain ratings of outfits and clothing from volunteers, as well as other subjective data that can be used to personalize recommendations
- Explore the Polyvore [Polyvore, 2017] dataset, which consists of fashion items (mainly clothing, shoes and accessories) organized in a tree of categories, and outfits composed by Polyvore users from images of the items
- Enable to build in and test machine learning models through the user interface of the application, using the data it contains

The application provides user interfaces for browsing and managing data obtained by automatic crawling from Polyvore and stored in a MySQL database. We have built in additional features for filtering outfits based on predefined criteria, or simple rules, that we will describe in more detail in Section 3.2. The purpose of this filtering is to clean some of the errors in the dataset and keep only correctly defined outfits for our later use. The application is also able to randomly generate new outfits, which are correct according to the same criteria.

The data we have obtained so far lack reliable information regarding which outfits are good or bad, that could be used in supervised learning algorithms. This inspired us to develop a new functionality for obtaining ratings of outfits and clothing from users based on their personal style preferences. Such subjective data about users' personal style can be used later for testing machine learning algorithms. The application was created with the possibility to build in such algorithms, mainly for generating outfits and recommending clothing items, and to test them directly via a user-friendly interface.

The application can run locally, however, to be shared with other users, its test version has been deployed on a cloud computing platform Amazon AWS. It uses the services of a virtual server EC2 (Elastic Compute Cloud) and Elastic Beanstalk to manage the infrastructure. These services also provide automatic scaling of computing resources according to the needs of the application. The virtual server also allows installation of additional libraries, e.g., for machine learning. The test version of the application, which is used to obtain ratings from volunteers, is currently available at `http://dressandgo.moniq.sk`.

3.1 Features

There are two categories of users in the application, with different available functionalities:

Regular users can rate a randomly selected sequence of outfits through a single-page application

Admin/Staff users can browse the whole data set, filter and randomly generate outfits, add clothing items to a virtual wardrobe, and rate outfits and items

Features of the application include:

- User registration and login.
- Browsing the dataset via paginated catalogs of outfits and clothing items. This dataset consists of fashion items (mainly clothes, shoes and accessories – containing image, category and text description) and outfits composed by Polyvore users from images of the items, along with links to the original items. Clothing items are organized in a tree structure of categories. Users can filter items by a category and all of its subcategories. It is also possible to search for items in the catalog.
- Filtering outfits according to a manually designed schema, described in more detail in Section 3.2.2. The purpose of this filtering is to clean up automatically scraped data, which does not always consists of complete outfits, or may contain incorrectly categorized items. This process of cleaning up and standardizing outfits also includes removing duplicate clothing and non-clothing items that sometimes also occur in the outfits.
- Generating random outfits according to the schema, using prior probability of clothing types obtained from the data, as described further in Section 3.2.5.
- Rating a randomly selected series of outfits. These outfits can be chosen either from those obtained from Polyvore or from a set of randomly generated outfits. The rating process is performed through a single-page application optimized for an efficient user experience.
- Rating a randomly selected sequence of clothing items filtered by type through a similar single-page application.
- Adding clothing items to a user's virtual wardrobe. The user can then view and browse all items in her virtual wardrobe.
- Selecting clothing items that the user likes from the catalog.

Later on, regular users should be able to access also other features, such as browsing the catalog of clothing, adding items to the wardrobe, and so on. After implementing machine learning models into the application, volunteers will be able to test the models in order to evaluate the obtained results.

3.2 Filtering and generating outfits

3.2.1 Clothing types

Women's fashion items in the Polyvore dataset are organized into 104 categories. These categories form a tree-like structure shown in Figure 3.1. Each item is assigned to a single category. We will select only several base categories in order to define a simplified structure of an outfit based on these categories, or *clothing types*.

To propose a set of clothing types, we will use the observation that some categories have an interchangeable function in an outfit and usually a person wears only one item from such a group of categories. For example, a person doesn't usually wear pants and a skirt at the same time, or two pairs of shoes, and so on. There are plenty of exceptions from the rules that we will define, but to move forward, we will need to standardize and define a simple outfit formula with regards to categories of the items in an outfit, even though this may exclude many wearable outfits.

For this purpose, we define clothing types based not only on similarity of the categories they contain, but also on the function of items from these categories in an outfit, or their position on the body. The proposed set of clothing types and the categories they contain is shown in Table 3.1. Each mentioned category contains also all of its subcategories.

Clothing type	**Item (Super-)categories**
shoes	'Shoes'
bag	'Bags'
dress	'Dresses', 'Jumpsuits & Rompers'
top	'Blouses', 'Sweaters', 'Sweatshirts & Hoodies', 'T-Shirts', 'Tank Tops', 'Tunics'
bottom	'Skirts', 'Jeans', 'Pants', 'Shorts'
outerwear	'Outerwear', 'Cardigans'
hat	'Hats'
scarf	'Scarves'
necklace	'Necklaces', 'Charms & Pendants'
earrings	'Earrings'
wristwear	'Watches', 'Bracelets & Bangles'
ring	'Rings'
eyewear	'Eyewear'

Table 3.1: Basic clothing types and the categories they contain

We excluded categories that had very few occurrences in the outfits from the dataset, such as Activewear, Belts, Gloves, etc., although some of them form a separate clothing type. This is just a further simplification of the outfit definition.

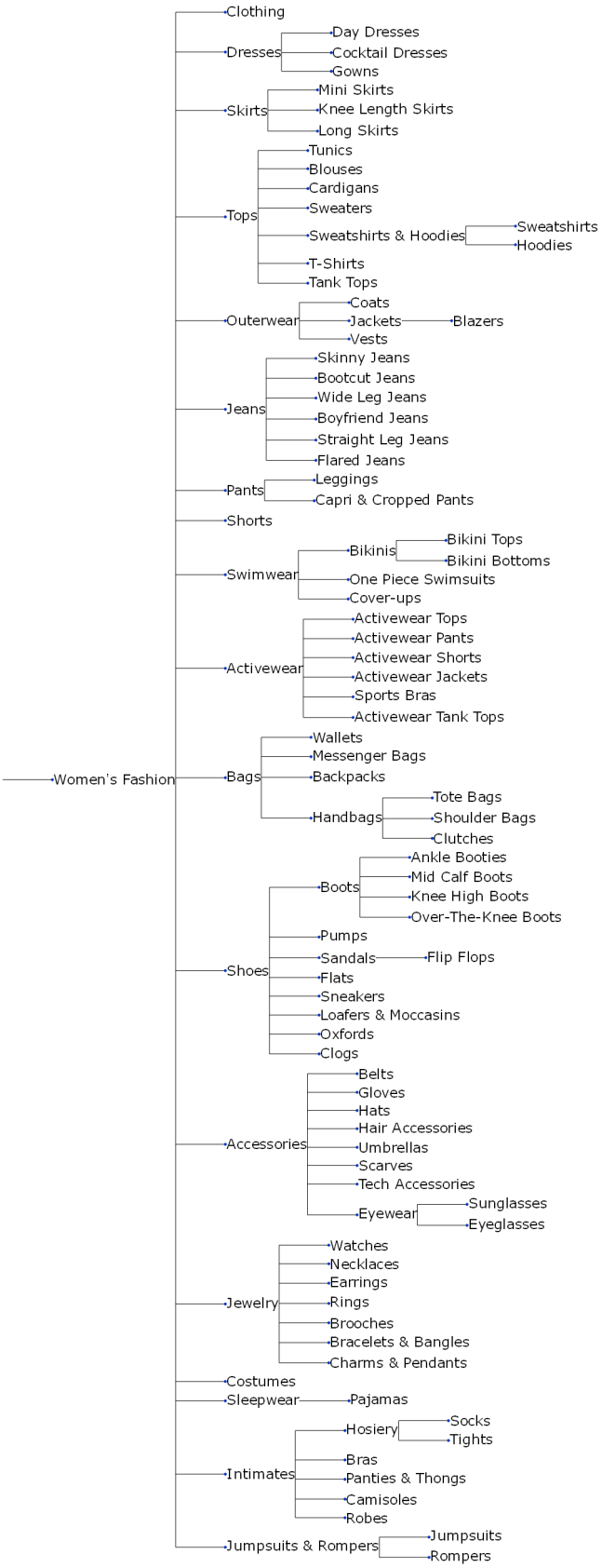


Figure 3.1: Tree of categories of women's fashion items

3.2.2 Basic structure of an outfit

Based on the clothing types proposed in the preceding section, we will define a set of simplified rules that describe which clothing types together form an outfit. This schema will be used for filtering out outfits that are non-standard in some way, and also for removing multiple items of the same type when filtering clothing in an outfit. It will also be used when generating new outfits, either randomly or using genetic algorithms, and would ensure that only "correct" outfits are generated.

We define the basic structure of an outfit as follows:

- An outfit must contain either **top** and **bottom**, or **dress**, but not both options.
- An outfit must contain one pair of **shoes** and one **bag**.
- An outfit can optionally contain at most one item of each type: **outerwear, hat, scarf, necklace, earrings, wristwear, ring, eyewear**.

We can use this structure to define a fixed-length vector representation of any outfit. Since an outfit can contain at maximum one item per clothing type, we can reserve several positions in the outfit vector to represent the item of each type. We will explore this in later chapters, when developing machine learning algorithms to evaluate the fashionability of an outfit represented by such a vector.

3.2.3 Filtering out incorrect outfits

If an outfit is not correct according to the set of rules from the previous section, it will not be used for collecting ratings from users and later evaluation using machine learning. This may happen when it does not contain all of the necessary items, or when they were incorrectly categorized by the outfit's creator.

If an outfit contains more than one item of a single type, we do not exclude the outfit, but rather use only the first one of the items when parsing the outfit, as described in the next section.

Out of the 2982 outfits that we have scraped from Polyvore in this batch, 1798 (60%) passed the filter according to our schema.

3.2.4 Parsing items in an outfit

If an outfit passes the filter, we proceed to parse the items in the outfit. By that we mean assigning for each possible clothing type the corresponding item, if present in the outfit.

Any item that is not contained in the categories or subcategories in Table 3.1 will be filtered out from the outfit. If there occur multiple items of the same type in an outfit, we remove all except the first one from the outfit.

This method filters and parses the items in an outfit mostly correctly, removing non-clothing objects (such as makeup) and duplicates, except when a certain item is incorrectly categorized. We can see several parsed outfits in Figure 3.2.

To display the items in an outfit conveniently to the user, we put their images in three columns:

Left: main items - dress, or top and bottom

Middle: outerwear (if present), bag, shoes.

Right: rest of the accessories, ordered head-to-toe.

This should aid user in visualizing the items as a complete outfit when rating them.

3.2.5 Randomly generating outfits

In the Polyvore dataset, most outfits tend to be highly rated and stylish. To provide the models also with negative examples of outfits, we can generate outfits randomly according to the rules described in Section 3.2.2. For each of the clothing types that were defined to be necessary, we select a random item of that type from our database of clothing. We include each of the optional item types with the probability of that type occurring in an outfit. This probability is calculated based on the available data as the fraction of outfits containing the item type from all the outfits (Table 3.2).

Clothing type	**Probability**
outerwear	25.3%
hat	8.8%
scarf	2.9%
necklace	18.3%
earrings	33.8%
wristwear	20.4%
ring	12.6%
eyewear	27.2%

Table 3.2: Probabilities of optional clothing types occurring in an outfit

We also estimated based on the data that the combination of top and bottom occurs in approximately 50% of the outfits, whereas a dress occurs in the remaining 50%. These are then the probabilities of selecting each of the two mutually exclusive options.

(a)

(b)

(c)

Figure 3.2: Original image sets or outfits (left); parsed and filtered items from each outfit (right)

(a) (b)

(c) (d)

(e) (f)

Figure 3.3: Examples of randomly generated outfits

In Figure 3.3, we can see that outfits randomly generated in this way often consist of clothing combination that are possible, but not appropriate or stylish to wear together. We have not so far included into these rules any information about colors, style and other useful attributes of clothing, that could influence their combination. We would like to improve upon that by using for example genetic algorithms to optimize towards better outfits.

New outfits are stored in the database so that users can rate them along with manually created outfits. We will need to create a tool for measuring whether an outfit is good and we will attempt to obtain such measure using supervised learning from the data. For that, we need examples of good outfits, as well as bad outfits. Later on, when using genetic algorithms, this measure can be used as a fitness function.

For collecting user ratings, we now have 3960 outfits in total, of which 1798 (45%) were designed by Polyvore users and 2162 (55%) have been randomly generated.

3.3 User interface

The user interface of the application was designed with focus on user experience and design. It uses Elite Admin design theme [Joshi, 2016], which was developed for complex administrator interfaces and contains many different components. Some of the front-end parts of the application use JavaScript and jQuery to retrieve data from the server through AJAX (Asynchronous JavaScript and XML) without the need to reload the whole web page.

The interface for regular users is simple and straightforward. After registration, the user can enter a single-page application to rate the outfits (Figure 3.4). When the user selects a rating choice, the data is sent to the server and the next outfit is retrieved in the background through AJAX. It is then displayed, without refreshing the whole page, in order to speed up the process of data collection. Users can in this way rate several hundred outfits within a few minutes. The progress bar of the rating process is also shown and updated dynamically. A similar application has been provided also for rating of clothing items, filtered by type.

The Admin/Staff user interface is more complex. It contains multiple different pages and catalogs with pagination, and menu on the left contains a tree structure of the categories (Figures 3.5, 3.6, 3.7).

3.4 Extending the application

The application can be further extended, for example by building in machine learning algorithms to recommend clothing items or outfits.

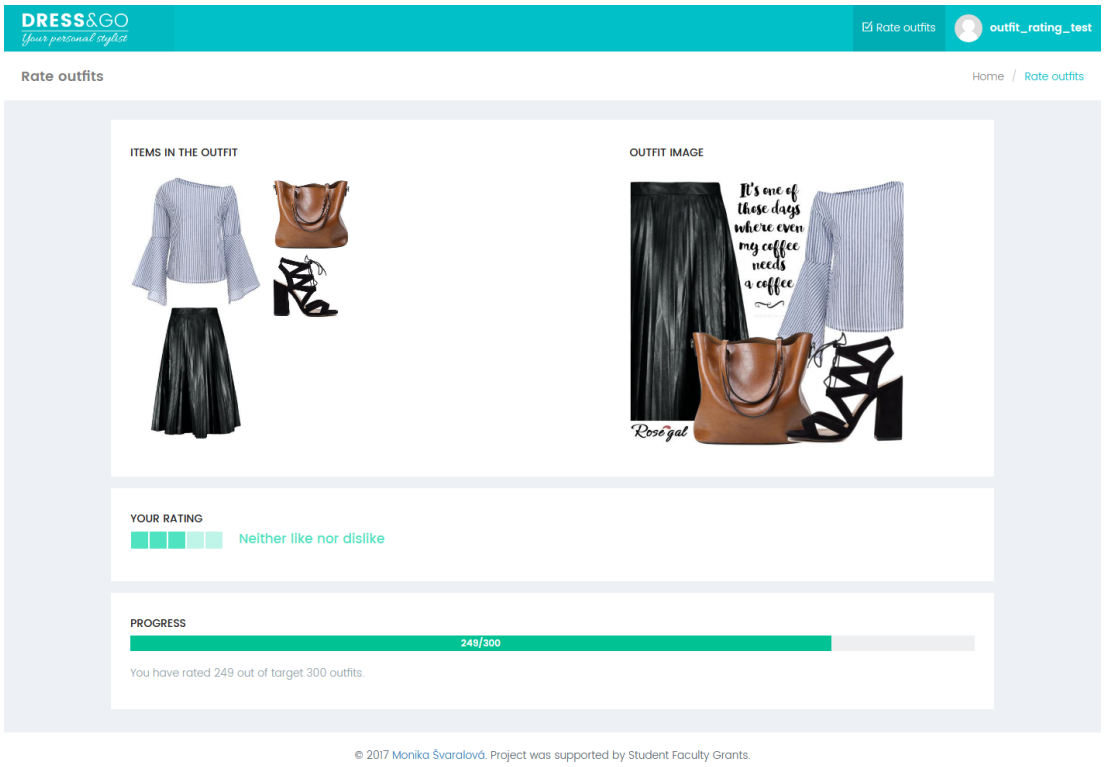


Figure 3.4: Single-page application for rating outfits. On the left, there are fashion items that have been filtered and parsed from the manually created collage on the right

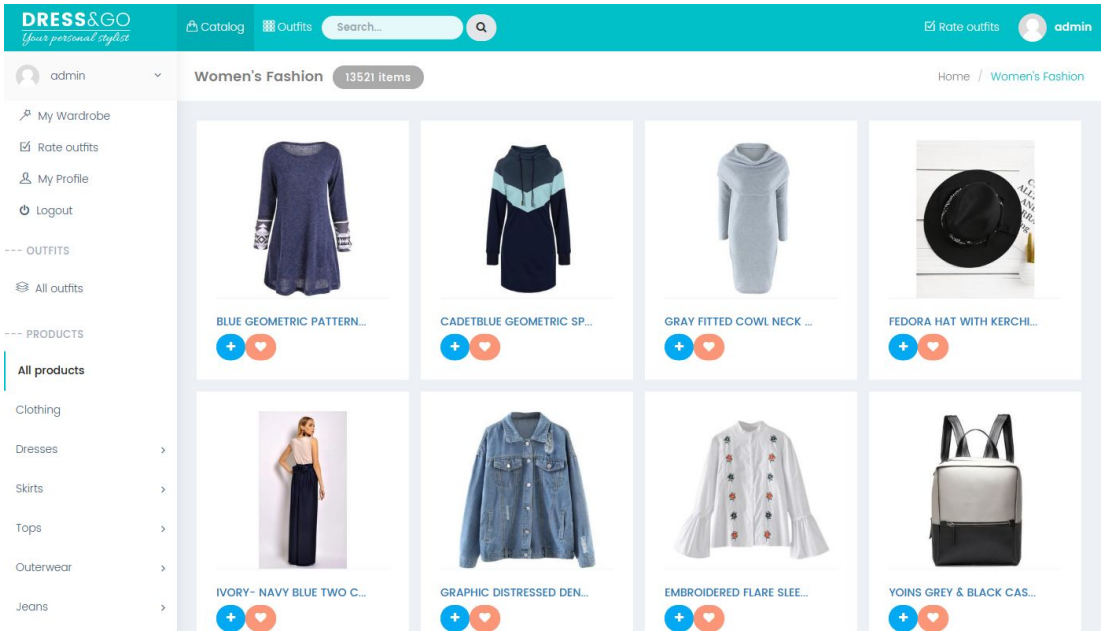


Figure 3.5: Admin interface for browsing the dataset and testing various features. This page contains a catalog of items, a menu with the tree structure of categories, and there is an option to add each item to the user's wardrobe or to "like" it

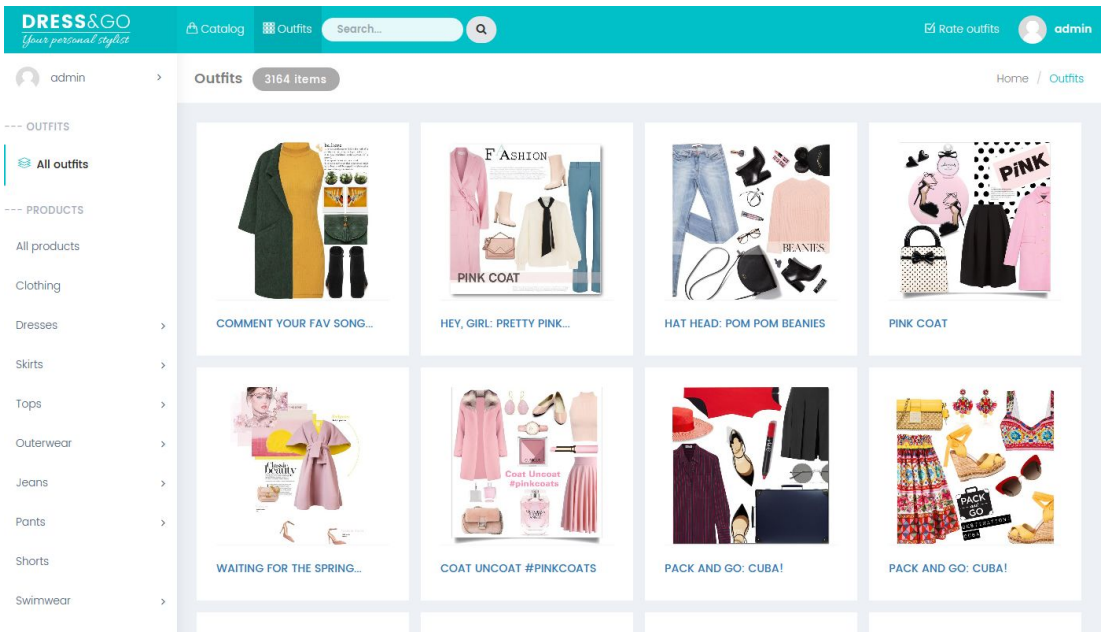


Figure 3.6: Admin interface, catalog of outfits

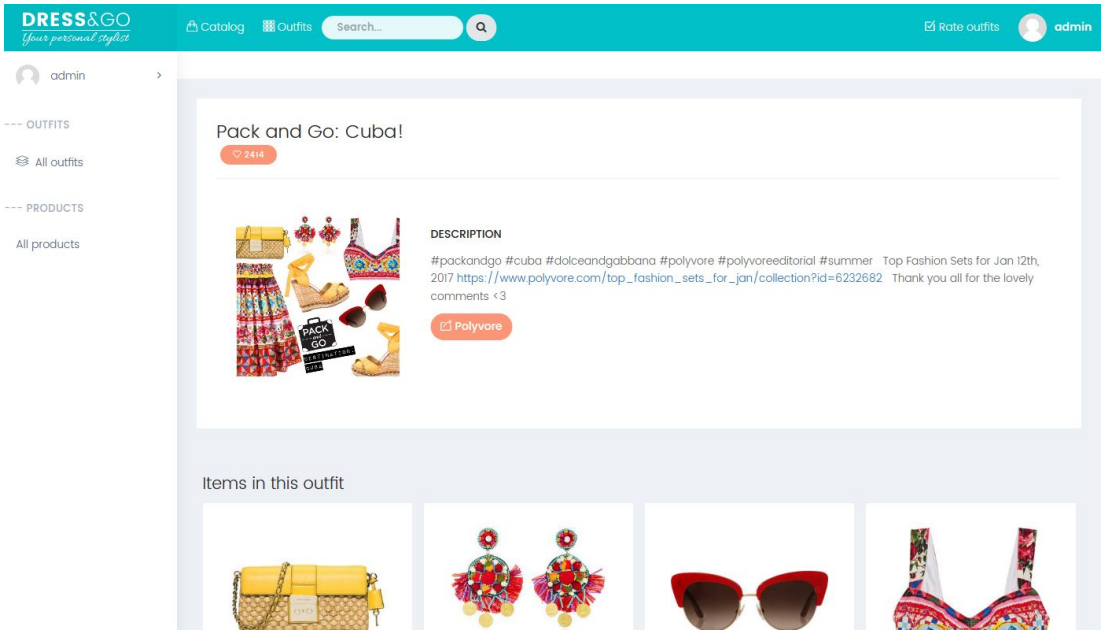


Figure 3.7: Detail page of an outfit, which contains also a list of all the items from the outfit. Similarly, a detail page of an item shows all the outfits which contain that item.

3.4.1 Target functionality

Other features that could be added to the application, and might be interesting to users, are the following ones:

- Enhanced virtual wardrobe, where a user could easily add, upload or search for items that she already owns
- Shop directory which recommends products that the user may like based on items in the user's wardrobe
- *Would it be wearable?* When considering to buy a new piece of clothing, for example, in the shop directory, the user could check whether it fits well with other pieces in her wardrobe, by previewing what outfits she could wear with this new piece
- Automatic and manual creating of outfits from items either in the user's wardrobe, or the whole catalog; browsing outfits published by other users

3.4.2 Machine learning methods

Since the application is implemented in Python using Django framework, it is possible to build in machine learning algorithms using Python libraries (such as Scikit-learn, TensorFlow, Keras, etc.) directly into the application. When built into the application, the machine learning methods can receive data using Django's database access API for a simplified and higher-level access to the database.

The data can also be retrieved by directly querying the database of the application, in which case the methods do not have to run under the instance of the Django application. Such trained models can be later built into the application for testing and visualizing the results.

4. Color clustering

Color is one of the most important properties of clothing and it also influences how we want to combine multiple pieces of clothing into an outfit. Therefore, one of the goals of the thesis is to automatically find reliable information about the main colors in an outfit or a piece of clothing from its image.

4.1 Finding dominant colors in an image

K-means clustering algorithm is suitable for finding dominant colors in an image. The set of data points for the algorithm are the color values of each pixel in the image, e.g., in the RGB color space. After finding clusters of color values using k-means, we will consider their centroids to represent the dominant colors of the image. The algorithm needs an input parameter k, the number of clusters. Since we do not know the number of main colors in an image in advance, we will try several values of k. We will then use a cluster validity metric to evaluate the cluster assignments in order to automatically select the best number of clusters k for each image.

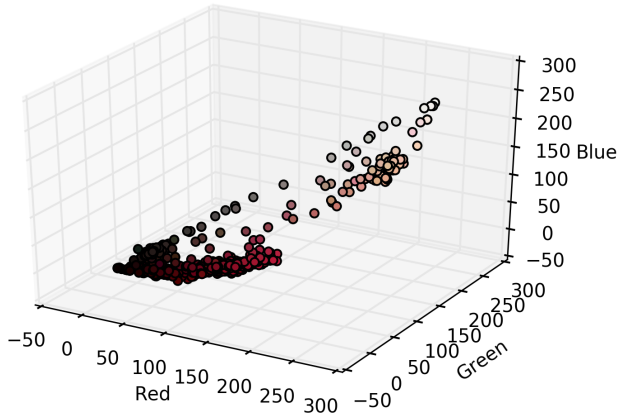
4.1.1 Data preparation

First, we need to remove the background from the images, because we would like to consider only the colors of clothing and other related objects. Most of our images contain products on a white background (with the exception of real life photos which are sometimes added to the outfit images). Therefore, we can discard all white pixels, and those that are almost-white, to remove also some of the JPEG artifacts and almost-white pixels at the borders of objects in the image. In this context, we choose the cut-off value so that we do not remove parts of white clothing items, which are usually darker than the background.

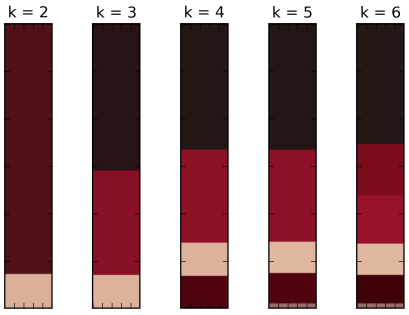
Since the outfit images are of dimensions 600×600px, it would be inefficient to run the k-means algorithm on the set of all pixels. We will select a random sample of pixels from the image as our set of points to be clustered. We can also remove the background during sampling, by discarding all the pixels that are of background color, which is more efficient than removing it in advance by checking all the pixels in the image. Instead of sampling, we could also resize the image, but this would result in averaging colors from the neighboring pixels. Since some of the outfit images contain fine-grained features such as black text on a white background, this would result in gray pixels which did not occur before.

In total, we will sample 1000 pixels from each image, which gives us a set of data points in the three-dimensional R,G,B space for each image. In Figure 4.1b, we can see a scatter plot of a sample of pixels from the image of a dress shown in Figure 4.1a. First, we will test the color clustering algorithm on several images of clothing items and whole outfits.

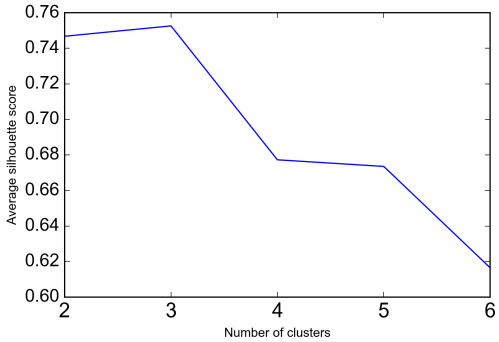
(a) Example image



(b) Color space visualization. Points represent the sample of pixels from the image. The color of each point in the scatter plot is the color of the original pixel.



(c) Dominant colors extracted from the image for different numbers of clusters k. The size of each color area reflects the percentage of points belonging to that color's cluster.



(d) Silhouette scores for different values of k

Figure 4.1: Using k-means clustering on an example clothing image

4.1.2 Results

Clustering the extracted data points using k-means yields the centroids of the clusters, which we will consider to be the dominant colors in the image. We will also find the relative size of each cluster, as the proportion of points assigned to the cluster. We will evaluate this for several values of $k \in \{2, ..., 6\}$ (Figure 4.1c).

A similar approach can be used for outfit images (Figure 4.2), where we will focus on finding the main colors in the whole outfit, so we could see what combination of colors was used. We can use this information to find colors commonly used together in outfits. However, sometimes slightly different colors are assigned together to form a single cluster, which creates a centroid with a color mixed from original ones. This happens especially for outfits with colors that occupy only a small area in the image.

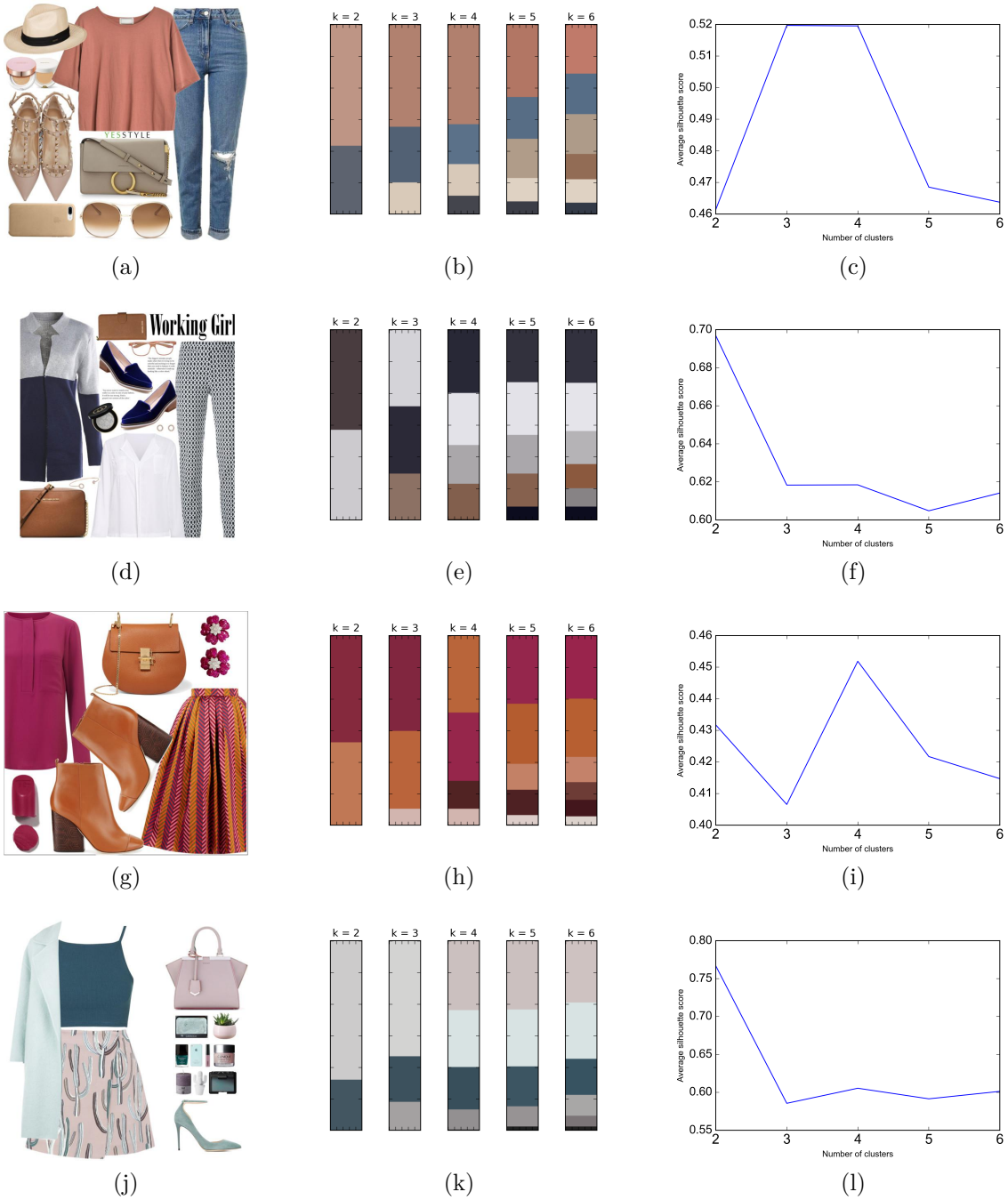


Figure 4.2: Results of color clustering for several images of outfits. For each image on the left, the resulting color clusters for different values of k are shown in the middle, on the right, there are the silhouette scores, again for different values of k

4.1.3 Optimal number of clusters

We would like to automatically determine the most optimal number of clusters k in order to find the main colors in an outfit. There are several metrics which evaluate the quality of cluster assignment, which we can use to evaluate the resulting clustering for various values of k. However, many of them require manually assigned labels to the data points.

We used mainly the silhouette metric [Rousseeuw, 1987]. It measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). It is computed as the mean silhouette coefficient over all the samples.

The silhouette coefficient $s(i)$ of a sample i is calculated using the mean intra-cluster distance $a(i)$ and the mean nearest-cluster distance $b(i)$ for the nearest cluster of which i is not a member:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{4.1}$$

Its values range from -1 to 1. Higher values indicate better clustering. Silhouette coefficient is only defined for the number of clusters k such that $2 <= k <= n_{samples} - 1$. However, we sometimes might want to have only a single cluster, such as when clustering an image of clothing, which is of a solid color.

We can see the results of this metric used for clustering in Figure 4.1d. While the coefficient works well in the case of some images, it is not a completely reliable measure. When the colors are similar but visually distinct, for example, similarly light or dark, like in Figures 4.2d and 4.2j, it prefers the lower number of clusters. In our data set of outfits, it usually has the highest value when $k = 2$.

4.2 Finding color relationships in outfits

We can now attempt to find relationships between the main colors extracted from the outfits. We would like to find some associations between the colors worn together, such as which combinations match and are stylish. For this purpose, we will use our whole dataset of 3000 Polyvore outfits and we will select the number of clusters in each outfit image according to the highest value of the silhouette score.

The clustering of each image I results is a set of $k_I \in \{2, ..., 6\}$ colors, each from the color space $\{0, ..., 255\}^3$. We would like to reduce the number of colors, therefore we lower the color depth to 4 levels per each channel R, G, B. This result in mapping of the colors represented by the cluster centers to a set of $4^3 = 64$ colors. We have found that the most frequent colors in the dataset are white, black and gray, followed by some of the other neutral colors. Such a low number of distinct colors allows also to illustrate their pairwise relationships, e.g., in a correlation matrix.

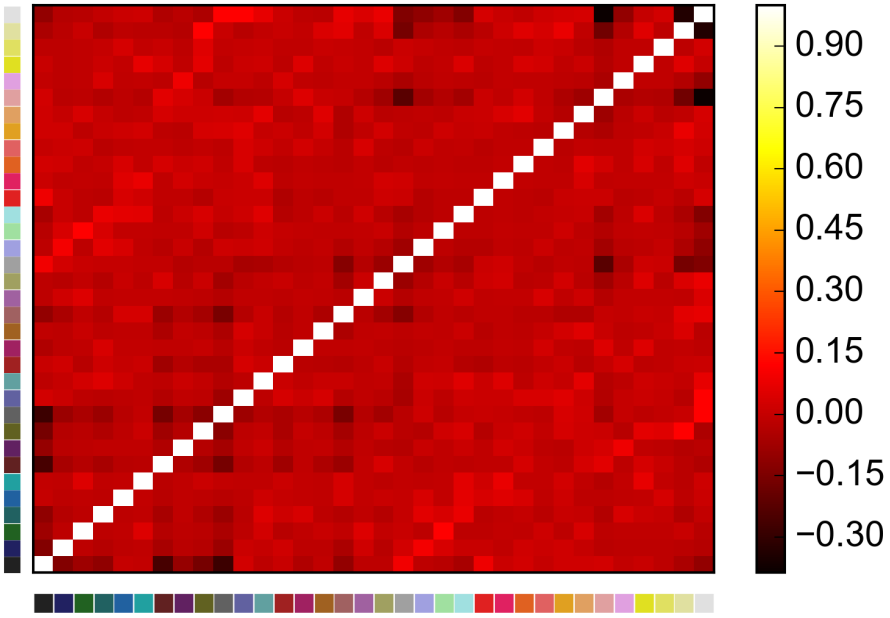


Figure 4.3: Correlation matrix of the pairs of colors occurring together in outfits, as extracted by color clustering. The colors are shown as labels, lighter value in the matrix means higher correlation between colors

To compute the correlation matrix, we will calculate pairwise correlations of color variables. The variable for each color means whether the color clustering of an outfit contains it as one of the dominant colors. In this case, we have 64 color variables as columns of a data matrix, with rows representing outfits. We will use Pearson's correlation coefficient between two random variables X_i and X_j at columns i, j:

$$\text{corr}(X_i, X_j) = \frac{\text{cov}(X_i, X_j)}{\sigma_{X_i}\sigma_{X_j}} = \frac{E[(X_i - \mu_{X_i})(X_j - \mu_{X_j})]}{\sigma_{X_i}\sigma_{X_j}} \tag{4.2}$$

where *cov* means covariance of variables. The correlation matrix is symmetric because $\text{corr}(X_i, X_j) = \text{corr}(X_j, X_i)$.

We can see the resulting correlation matrix visualized at Figure 4.3. Only the most common 34 colors are shown. Because the result of the color clustering is not always correct, it is difficult to see whether the correlation matrix shows meaningful relationships. However, it suggests that some color combination are worn more often together, and others are avoided. For example, color pairs that are dark, such as black and brown, or black and dark gray, do not correlate, as well as light color pairs, such as white and light beige. This may however be also caused by these similar colors getting put into the same cluster, which would make it difficult to show the relationship between them.

4.3 Clustering outfits into groups of similar colors

For illustration, we will also cluster a set of outfits into several groups by the whole color palette of their image. The resulting clusters could represent color trends, or commonly occurring color combinations. We will need to transform the outfits into a vector space in which those with similar color scheme would be close, e.g., by means of Euclidean distance. For this, we will use a color histogram of each image. This approach would also enable searching for images with a similar color scheme.

4.3.1 Color histogram

A color histogram represents the distribution of colors in an image. It is constructed by counting the number of pixels that have the respective color in each bin, or a fixed list of color ranges spanning the color space of the image. For an image represented in the RGB color space, this yields a three-dimensional histogram, each dimension consisting of a number of bins determined in advance.

4.3.2 Outfit clusters

We will transform each outfit image by flattening its three-dimensional color histogram into a vector. In this new vector space, images similar in color would be close to one another. We could use only the previously computed dominant colors, but this would lose some information about all the colors in the image, especially since the previous results of color clustering are not always optimal. In the vector space of outfit color histograms, we can use a clustering method such as k-means to find groups of similar outfits.

We decided to use 4 bins in the histogram for each of the R,G,B dimensions. In each color channel, bin $i \in \{0, \ldots, 3\}$ corresponds to intensities $\langle 64 * i, 64 * (i + 1) - 1 \rangle$. This results in $4^3 = 64$ distinct colors. Colors that are similar fall into the same bin in the histogram, and they are treated as one color. Instead of using absolute pixel counts, we divide them by the total number of non-background pixels in order to obtain percentage, because different outfits can cover differently large parts of their respective images.

We try this on a random sample of 500 outfits from our dataset. From each outfit image, we compute the color histogram vector. We then cluster the vectors into 50 clusters using k-means, to obtain on average 10 outfits per cluster. Several of the clusters along with the images closest to their centers are shown in Figure 4.4, which gives some indication of what the color scheme of each cluster is like. The results are similar even when using histograms with 6 or 8 bins in each dimension.

(a)

(b)

(c)

(d)

Figure 4.4: Clusters resulting from using k-means on color histograms of outfit images. We show four of the 50 clusters, each row corresponding to one cluster. The clusters are described by showing three outfit images closest to its centroid. To the right of each outfit are its extracted dominant colors, for illustration.

5. Image classification with convolutional neural networks

The natural approach to use on image data, which is the main form of information that describes a piece of clothing, are convolutional neural networks. In the past few years, they have been very successfully applied on many types of image processing and classification tasks.

In this chapter, we will test convolutional networks for image classification in the domain of fashion. We will compare the results of both small models trained from scratch, and large pre-trained models. The latter may be useful especially due to the small sizes of the datasets we will be working with, which contain only several hundred images per class. We will use images of clothing for various image processing and classification tasks, such as feature extraction and finding similar items, classification into fine-grained clothing categories, and finally, we will try whether a model could learn a user's personal style from a small supervised dataset.

For the implementation, we will be using Keras [Chollet, 2015] with TensorFlow [Abadi et al., 2015] in the backend. These are deep learning frameworks containing high-level interfaces for training even very complex convolutional networks, and support GPU acceleration.

5.1 Pre-trained models

Recent advances in deep learning brought about large convolutional network architectures trained on datasets such as ImageNet [Deng et al., 2009], which contain millions of images manually labeled into 1000 classes. These new architectures, along with improvements in computational efficiency, achieved unprecedented results in image classification of these datasets. Pre-trained weights of these networks have been made available for download and can be used for transfer learning.

Transfer learning focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, a convolutional network that has been trained on a large general dataset such as ImageNet, had learned convolutional filters that can be useful for various image classification tasks. The first few convolutional layers contain more general image features, while the last ones have more specialized features to the particular task.

To use pre-trained convolutional networks for transfer learning, we could extract outputs from a particular layer of the network and train a new classifier on the extracted features. Another option would be to retrain also a few of the network's higher-level convolutional layers in order to fine-tune their weights specifically for our task. Using a pre-trained network instead of training a whole

network from scratch can be helpful in improving generalization from a small dataset, and can be less computationally expensive. In our experiments, we will be using two models of these large pre-trained convolutional networks, which are described below.

5.1.1 VGG

VGG16 (Visual Geometry Group, 16-layer architecture) [Simonyan et al., 2014] is a convolutional network with five blocks of convolutional layers, each followed by max-pooling. Each convolutional unit has a 3×3 receptive field. Max-pooling is done from 2×2 units. On the top of the network are 3 fully-connected layers, 4096 units each. All hidden layers use ReLU activation function. A diagram of the network is shown in Figure 5.1. This model achieved 7.5% top-5 validation error in classifying the 1000 image classes of the ImageNet Large Scale Visual Recognition Challenge dataset.

5.1.2 Inception architecture

Inception v3 architecture [Szegedy et al., 2015] was based on several new ideas to scale up convolutional networks. For example, it reduces the number of parameters by factorizing convolutions with larger spatial filters (e.g. 5×5) into a mini-network of convolutions with 3×3 filters. A schematic diagram of the network is shown in Figure 5.2. This model achieved 5.6% top-5 and 21.2% top-1 error on the validation set of the ImageNet Large Scale Visual Recognition Challenge dataset. An ensemble of four of these models achieved 3.58% top-5 error.

5.2 Feature extraction

In order to extract relevant features from our images, we will use the outputs from a hidden layer of the pre-trained Inception v3 network. These so-called "bottleneck" features are the activations in the last layer before the final fully-connected layer that does the classification. They are a vector of size 2048 for each input image, and we can train a multi-layer neural network or other machine learning model using these image representations as their input. Because computing these bottlenecks takes some amount of time, we pre-compute and save the computed features for each image in our dataset.

The bottleneck layer has been trained to output a set of values from which a classifier can distinguish between the target classes. It has to be a meaningful and compact summary of the images. The features needed to distinguish between the 1,000 classes of the ImageNet dataset on which the classifier was trained are often also useful to distinguish between new classes of objects, such as clothing items in our example. Also, ImageNet contains several clothing-related classes, so the models may contain even more relevant features.

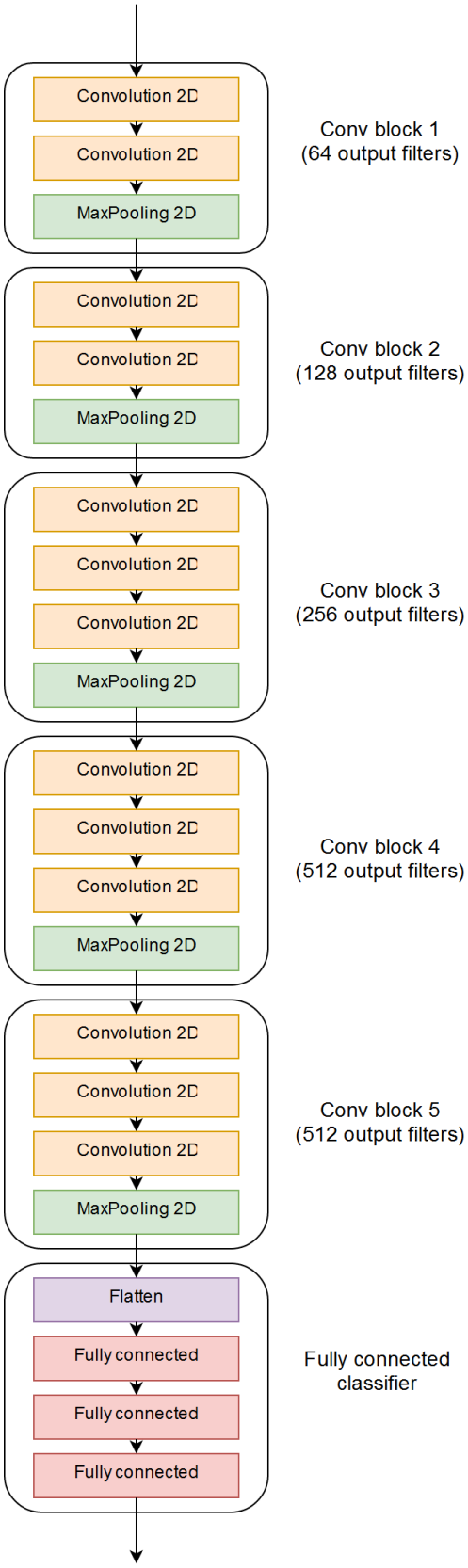


Figure 5.1: Simplified diagram of the VGG16 convolutional network

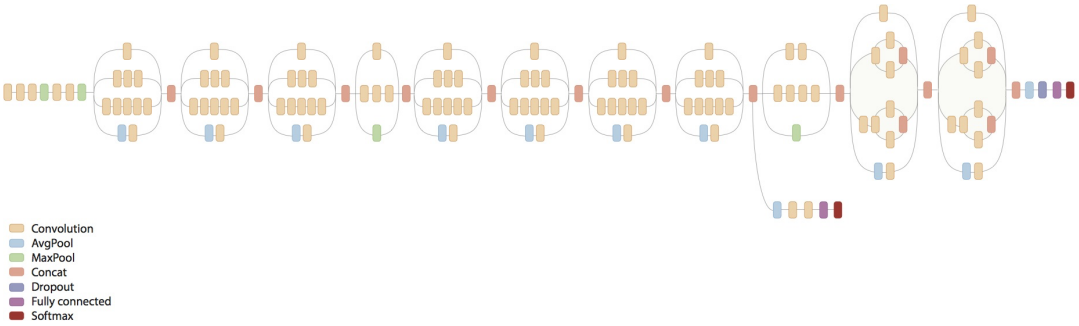


Figure 5.2: Schematic diagram of Inception v3 architecture. Image reproduced from [*Inception in TensorFlow*]

To explore the representations of our images extracted from the bottleneck layer, we will first visualize these representations using *t-SNE embedding*. We would like to see whether they contain relevant information, by exploring whether items that we expect to be similar would end up close in the embedded feature space.

5.2.1 t-SNE embedding

t-SNE (t-Distributed Stochastic Neighbor Embedding) [Maaten et al., 2008] is a machine learning algorithm for dimensionality reduction. It is well-suited for embedding high-dimensional data into a space of 2 or 3 dimensions in such a way that similar objects are modeled by nearby points and dissimilar ones by distant points. Visualizing the scatter plot of the resulting embedding allows us to see clusters of data points that are near each other in the original space.

First, t-SNE constructs a probability distribution P over pairs of high-dimensional objects from the set of N objects $\mathbf{x}_1, \ldots, \mathbf{x}_N$. The probabilities p_{ij} are proportional to the similarity of objects $\mathbf{x}_i$ and $\mathbf{x}_j$, so that similar objects have a high probability of being picked, whilst dissimilar points have an extremely small probability of being picked.

The similarity of datapoint $\mathbf{x}_j$ to datapoint $\mathbf{x}_i$ is the conditional probability, $p_{j|i}$, that $\mathbf{x}_i$ would pick $\mathbf{x}_j$ as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at $\mathbf{x}_i$:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)} \tag{5.1}$$

The variance of each Gaussian σ_i^2 is set in such a way that the perplexity of the conditional distribution equals a predefined value. The corresponding σ_i is found using binary search. As a result, the variance is adapted to the density of the data: smaller values of σ_i are used in denser parts of the data space.

The joint probabilities p_{ij} in the high dimensional space are then defined as symmetrized conditional probabilities, that is:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{5.2}$$

Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map, and it minimizes the Kullback–Leibler divergence [Kullback et al., 1951] between the two distributions with respect to the locations of the points in the map.

t-SNE aims to learn a d-dimensional map $\mathbf{y}_1, \ldots, \mathbf{y}_N$ (with $\mathbf{y}_i \in \mathbb{R}^d$) that reflects the similarities p_{ij} as well as possible. To this end, it measures similarities q_{ij} between two points in the map, $\mathbf{y}_i$ and $\mathbf{y}_j$, using a very similar approach. Specifically, q_{ij} is defined as:

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k \neq m}(1 + \|\mathbf{y}_k - \mathbf{y}_m\|^2)^{-1}} \tag{5.3}$$

A heavy-tailed Student t-distribution (with one degree of freedom, which is the same as Cauchy distribution) is used to measure similarities between low-dimensional points in order to allow dissimilar objects to be modeled far apart in the map.

The locations of the points $\mathbf{y}_i$ in the map are determined by minimizing the Kullback–Leibler divergence of the distribution Q from the distribution P, that is:

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{5.4}$$

The minimization with respect to the points $\mathbf{y}_i$ is performed using gradient descent. The result of this optimization is a map that reflects the similarities between the high-dimensional inputs well.

5.2.2 Visualization of the dataset using t-SNE

In order to explore the representations of images extracted from the bottleneck layer of the pre-trained Inception v3 network, we visualized their two-dimensional embeddings using t-SNE. We plot each image at its computed position in the plane. In Figure 5.3, we can see the embedding of a sample of 3000 women's clothing images from our dataset. In Figure 5.4, an embedding of the whole set of 13500 images is shown, along with manually annotated main clusters that are visible in the embedding.

We can clearly see clusters corresponding to some of the clothing types. Within a cluster, the images are also distributed according to color, shape or pattern. They are sometimes distributed also based on some extraneous information, such as the angle from which the product has been photographed. We can see this

in the cluster of shoes, where shoes photographed from the same angle are usually closer together. The visualization shows that the extracted bottleneck layer contains information relevant to clothing and their classification, including colors, shape, and so on. This may be also be the case because some clothing classes have been present in the ImageNet dataset on which the model was trained, although they are not as fine-grained as the ones we will use.

5.2.3 Nearest neighbors

Since the t-SNE embedding only shows approximation of which data points are actually nearby in the high-dimensional space of Inception v3 bottlenecks, we also looked at the nearest neighbors of several images in this representation. The distance was computed using Euclidean metric. We can see in Figure 5.5 that the images closest to the source image are usually similar in color, material, shape, etc. This approach of finding neighbors in a suitable representation could be used to search and recommend similar items in any kind of clothing database, e.g., in an e-shop.

5.3 Classification of shoe images

We have first tested the effectiveness of convolutional networks on a fashion image classification task using a small sample dataset of a few fine-grained clothing types. We have scraped a separate dataset of shoe images classified into 9 subcategories (Athletic, Boots, Clogs, Flats, Loafers & Moccasins, Oxfords, Pumps, Sandals, Sneakers) from the Polyvore catalog. Each class contains only 360 images, which is considered very few for training a convolutional network to generalize well.

In Figure 5.6, a sample of images from each class is shown. We can see that some of the categories are very similar, for example sneakers and athletic shoes, or loafers-moccasins and flats. Some of the shoes could belong to multiple categories, but had to be labeled only as one, which could further complicate the classification task.

There has been some previous research on clothing type classification, recognition, or segmentation, as mentioned in Section 1.4.1. Convolutional networks have often been the most successful approach, sometimes combined with other machine learning and computer vision methods. We selected this task because it is more fine-grained than classification into the basic categories of clothing, and the classes are not always easily distinguishable. On the other hand, the images in our dataset are much cleaner than the ones in most other classification experiments, containing a professional photo of each item on a white background.

Each classifier was trained on mini-batches of training samples, minimizing the categorical cross-entropy

$$H(p, q) = -\sum_{x} p(x) \log q(x) \tag{5.5}$$

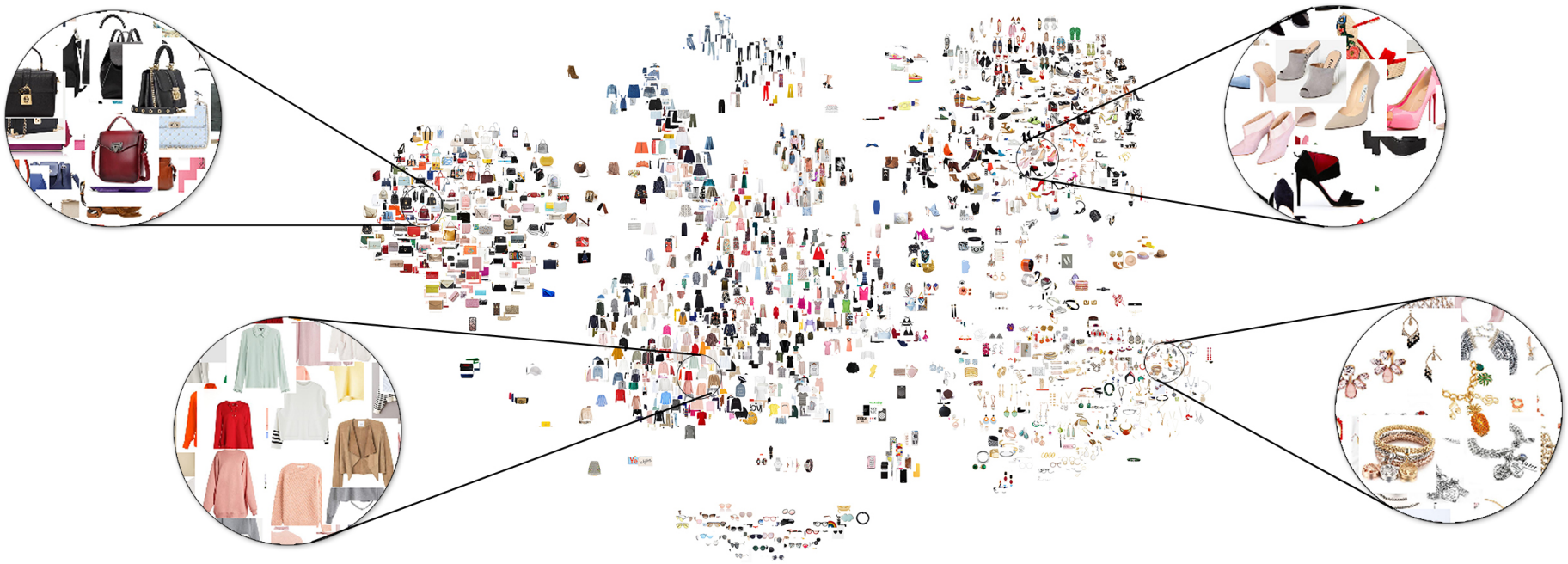


Figure 5.3: t-SNE embedding of a sample of 3000 images from the dataset of women's clothing. Each large cluster approximately corresponds to a clothing type (e.g., shoes, jeans, bags, tops, eyewear, etc.)

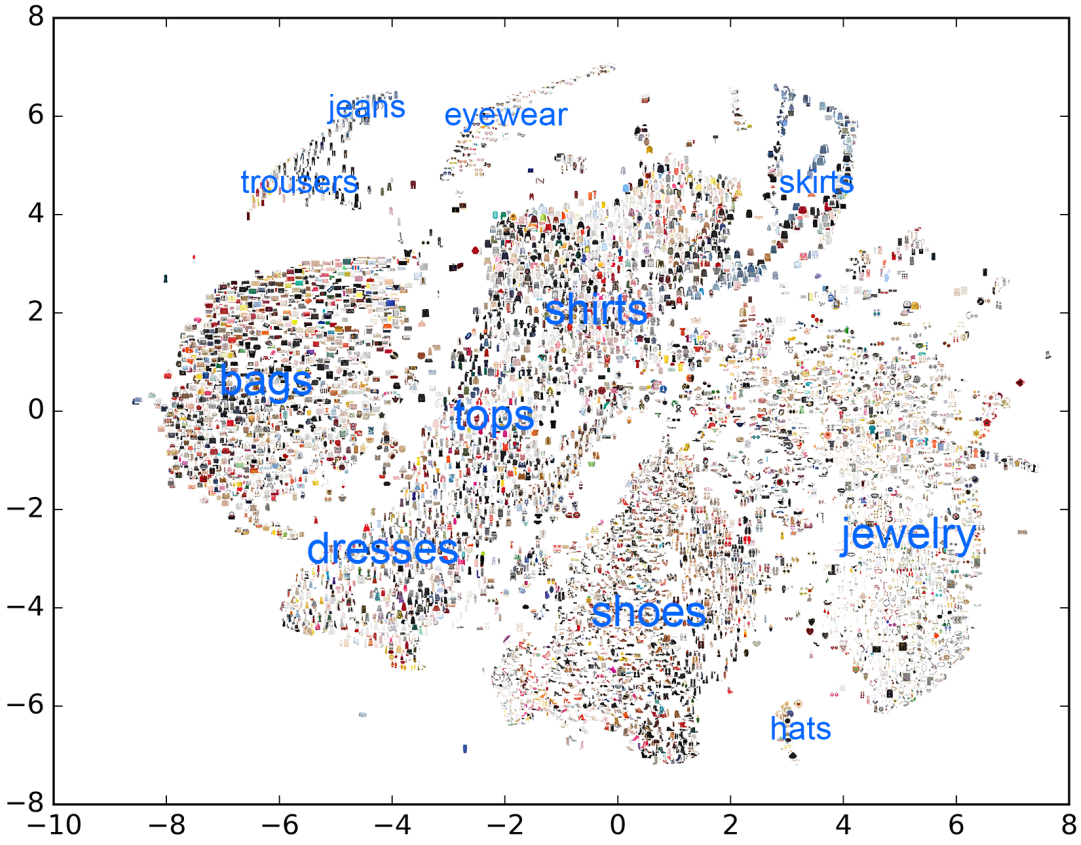


Figure 5.4: t-SNE embedding of the dataset of women's clothing, with annotated clusters corresponding to the majority type of clothing in the cluster

Figure 5.5: Five nearest neighbors in the high-dimensional Inception v3 bottleneck representation to the image on the left. The distance of each neighbor to the source image is shown.

(a) athletic

(b) boots

(c) clogs

(d) flats

(e) loafers-moccasins

(f) oxfords

(g) pumps

(h) sandals

(i) sneakers

Figure 5.6: Several examples from each of the shoe categories

(a) (b) (c) (d) (e) (f)

Figure 5.7: Examples of random transformations applied to the input images to improve generalization from the small dataset.

where p is the true distribution and q is the approximating distribution. The training was performed using Adam optimizer [Kingma et al., 2014], an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. It is a computationally efficient method suited for problems with large number of data or parameters. The evaluation of each classifier was performed using 10-fold cross-validation, reporting the average accuracy with 95% confidence intervals.

5.3.1 Training a small convolutional network from scratch

We first used a small convolutional network for image classification. Since we have relatively little data to learn the right features that distinguish between classes, random transformations were applied on the training images in order to improve generalization. Input images were originally all of the same size (300x300px), and have been resized to 100x100px in order to constrain the number of parameters.

The random transformations consisted of slight rotations, shifts in any direction, shearing, zooming in, and randomly flipping the image horizontally. A few examples of transformed images are shown in Figure 5.7. Keras library provides very useful high-level functions to achieve a variety of image transformations and generate randomly transformed images from the original dataset indefinitely.

The convolutional network was a simple stack of three convolutional layers with 3×3 convolutional filters and ReLU activations, each followed by a max-pooling layer from 2×2 units. Each convolutional layer had 32 output filters. They were followed by 2 fully-connected (dense) layers, first with 32 ReLU units and 50% dropout, and the output layer with a sigmoid activation function, with one output unit for each of the 9 target classes. We experimented with various parameter settings of the network, such as different layer sizes, activation function, dropout, number of training epochs, but it would be possible to tune them further.

An example code using Keras library to construct, train and evaluate such a network is as follows:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(100, 100, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Model	Mean CV accuracy	CI 95%
Baseline	*11.1%*	
Convolutional network	71.98%	66.15% – 77.81%

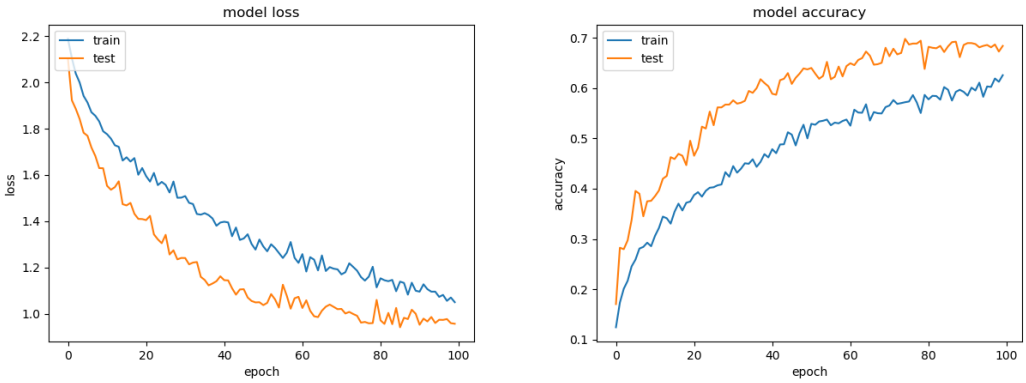
Table 5.1: Accuracy of classifying images of the shoe classes dataset using a convolutional network

```
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(n_categories))
model.add(Activation('sigmoid'))

model.compile(loss='categorical_crossentropy',
          optimizer='adam',
          metrics=['accuracy'])

model.fit(X_train, Y_train,
          batch_size=32, epochs=100, verbose=2)

score = model.evaluate(X_test, y_test)
```

The training was performed for 100 epochs, until the validation loss stopped decreasing. The resulting average cross-validation accuracy was 71.98% (Table 5.1). However, without using random input image transformations, the accuracy was only 30.31%. The progress of training and test accuracy and loss during training is shown in Figure 5.8. The higher accuracy on the test set compared to the training set can be explained by that the input images in the test set were not randomly transformed as those in the training set. These images may have been easier to classify, even though they did not occur in the training set. Confusion matrix of the test set predictions is shown in Figure 5.9. As expected, some classes proved more difficult to distinguish.

5.3.2 Retraining the Inception v3 network

We can further improve the results on the small dataset by using a pre-trained convolutional network. This is much faster than training the whole large net-



(a) Model loss (b) Model accuracy

Figure 5.8: Loss and accuracy progress during training of the convolutional network for 100 epochs

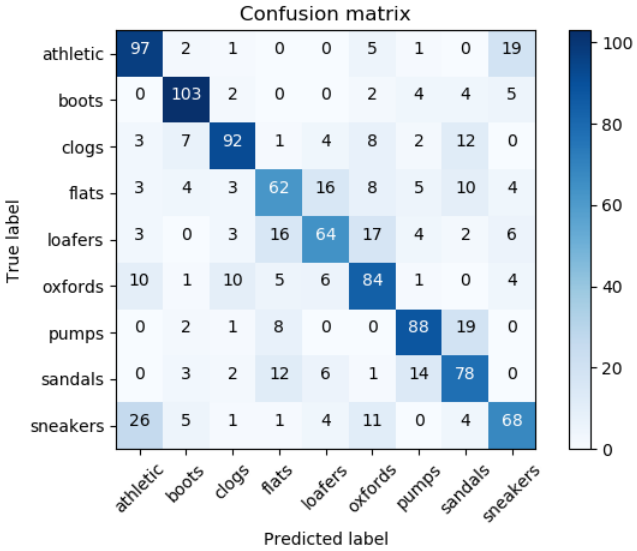


Figure 5.9: Confusion matrix of the convolutional network trained to classify shoe types

Model	Mean CV accuracy	CI 95%
Baseline	*11.1%*	
Neural network	78.59%	73.97% – 83.21%
Random forest	32.90%	27.82% – 37.98%
Support vector machine	35.46%	30.54% – 40.39%
k-nearest neighbors	68.37%	65.96% – 70.78%

Table 5.2: Comparison of model accuracies of classifying images of the 9-shoetypes dataset using bottleneck representations from Inception v3 network

work from scratch, and it utilizes the previously trained weights, in order to achieve better generalization even with the small number of images in our set. We extracted features of each image from the bottleneck layer of the Inception v3 network. This time, we don't transform the images randomly, rather, we compute the bottleneck representation for each original image in advance, because it is computationally expensive to run the model every time we introduce a new distortion.

The bottleneck representation is a 2048-dimensional vector. We used it as the input to a fully-connected neural network with one hidden layer of 64 ReLU units and 50% dropout. The output layer had a sigmoid activation function. We trained this model for 50 epochs, but the validation accuracy stopped increasing even before that, as can be seen from its progress during a single run of the model (Figure 5.10). We achieved an average cross-validation accuracy of 78.59%. The confusion matrix on the validation set is shown in Figure 5.11. We can see from the misclassified examples in Figure 5.12 that they are often very similar to the predicted class, sometimes even incorrectly classified in the original dataset.

Since the bottleneck features of the pre-trained network are a vector representation of each clothing item, we also tried other machine learning models using this representation as their input. Those, however, did not achieve as high accuracy as the simple neural network. We have previously seen from the t-SNE embedding and nearest neighbors visualization, that similar types of clothing tend to be nearby in the bottleneck representation, suggesting k-nearest neighbors classifier. It achieved average CV accuracy of 68.37%, which shows that also items from the same subcategory are usually close together. We have also tried support vector machines with 35.46% accuracy, and random forest resulted in 32.90% accuracy. Comparison of the results including 95% confidence intervals is shown in Table 5.2.

5.3.3 Retraining the last convolutional block and fully-connected layers of VGG16

Our final enhancement was inspired by [Chollet, 2016]. In addition to training a small fully-connected network on top of the bottleneck layer of VGG16, we fine-tuned also the last convolutional block of VGG16, in order to learn higher-level convolutional features that would be useful for this particular classification task.

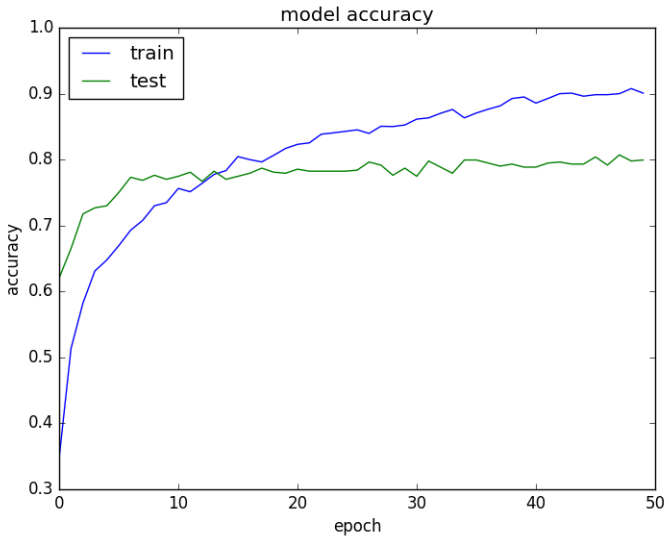


Figure 5.10: Accuracy progress during retraining of the Inception network for 50 epochs

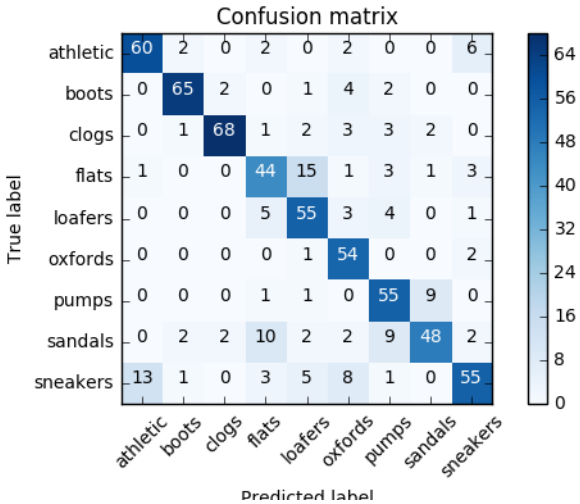


Figure 5.11: Confusion matrix of the retrained Inception v3 network

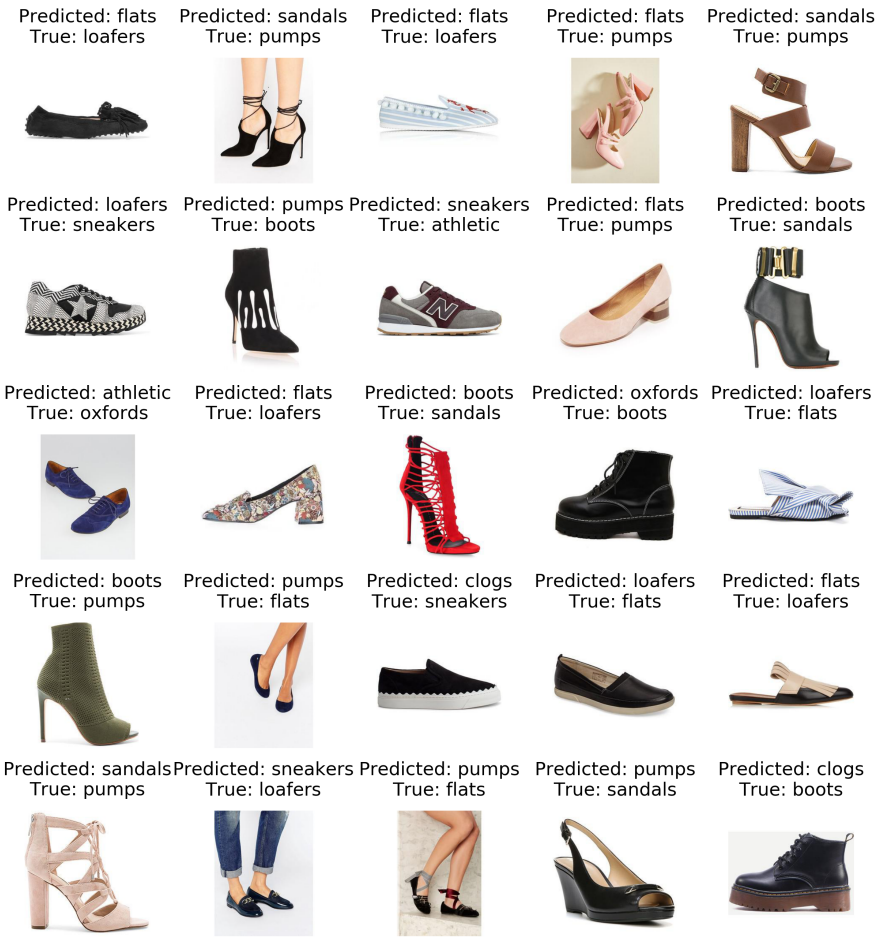


Figure 5.12: Incorrectly classified examples from the test set by the retrained Inception network

Model	Mean CV accuracy	CI 95%
Baseline	*68%*	
VGG16 top layers and conv block retraining	94.94%	93.74% – 96.13%

Table 5.3: Accuracy of classifying images from the shoe classification dataset using VGG16 model with retrained last convolutional block and a fully-connected neural network

The process is as follows: first, we train a small neural network upon the bottleneck features ("Flatten" layer) obtained from VGG16 for each image. This network is the same as in the previous experiment, with one hidden layer of 64 ReLU units. Then, we unfreeze the last convolutional block and train the joint network for 50 epochs, using stochastic gradient descent with a small learning rate. The fully-connected layers have to be trained first, otherwise, the gradient could be too large and we would lose the previous information in the convolutional weights. The weights in the first four blocks remain "frozen", which means that they do not change during training. The resulting network is shown in Figure 5.13.

After the first step of training only the top block of fully-connected layers, the accuracy is similar to the one we achieved using the Inception network. However, when retraining also the last convolutional block, we achieve the average cross-validation accuracy of 94.94% (Table 5.3). The combination of pre-trained features with the custom fine-tuning on our dataset seems to be very effective for image classification. The confusion matrix is shown in Figure 5.14, a few of the misclassified examples are shown in Figure 5.15.

5.4 Rating clothing according to user preferences

After the initial success in classifying fine-grained clothing types with convolutional networks, we turned to another task: whether we could use the same approach to classify a person's subjective opinion about a piece of clothing from a small labeled dataset. We tested predicting a person's rating of a clothing item solely from its image, in order to assess the use of such a method for personalized clothing recommendation.

The distinction between clothing images that a person likes and dislikes can be very nuanced, caused by a specific detail. We used the same dataset of shoes as in the previous experiments for better comparability of results. We collected ratings of 1000 shoe images from a single volunteer, with target labels consisting of only "like" or "dislike", to simplify the task to binary classification. A sample of images from each of the two classes is shown in Figure 5.16. In the dataset, 32% of images belonged to the "like" class, 68% to the "dislike" class. The classes were slightly imbalanced, so we also tried class weighting.

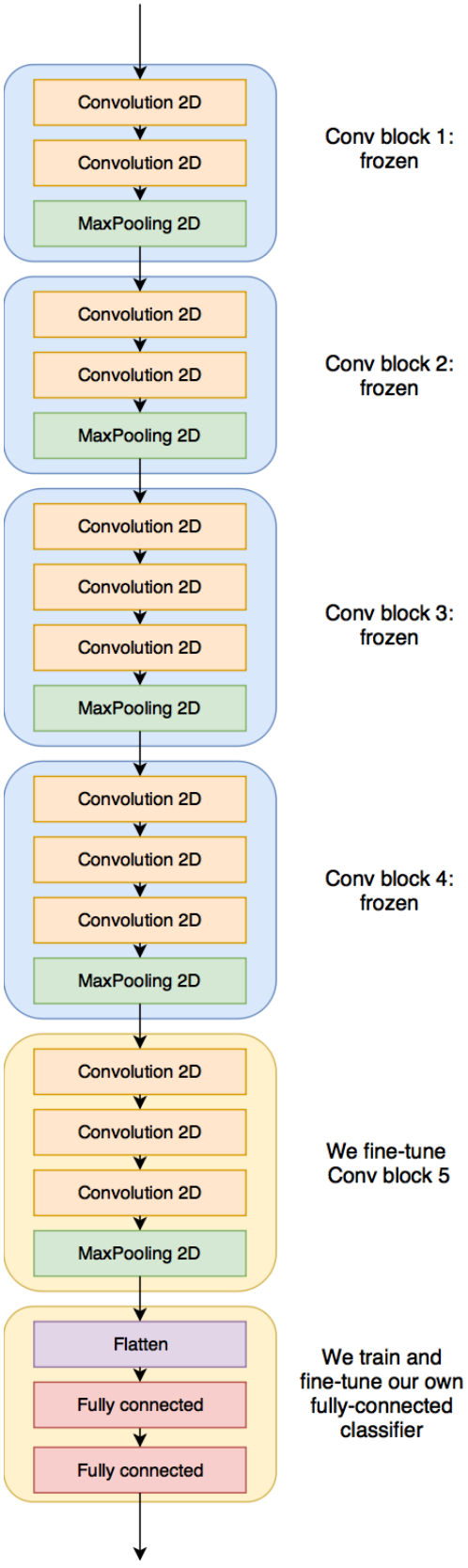


Figure 5.13: Retraining the VGG16 model – convolutional blocks 1-4 are frozen, we use a smaller fully-connected classifier instead of the original

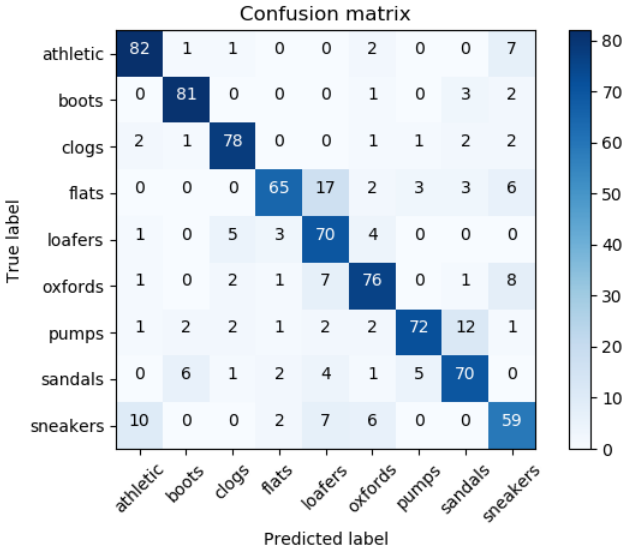


Figure 5.14: Confusion matrix of the retrained VGG16

Figure 5.15: Incorrectly classified examples from the test set by the retrained VGG16

(a) liked

(b) disliked

Figure 5.16: Examples from each of the rating categories by the user

Model	**Mean CV accuracy**	**CI 95%**
Baseline	*68%*	
Convolutional network	66.54%	61.46% – 71.62%

Table 5.4: Accuracy of classifying images of the user ratings dataset using a convolutional network

We used the same models and their parameters as in the previous experiments on shoe type classification. Instead of categorical cross-entropy, we minimized binary cross-entropy

$$H(p,q) = -\sum_i p_i \log q_i \tag{5.6}$$

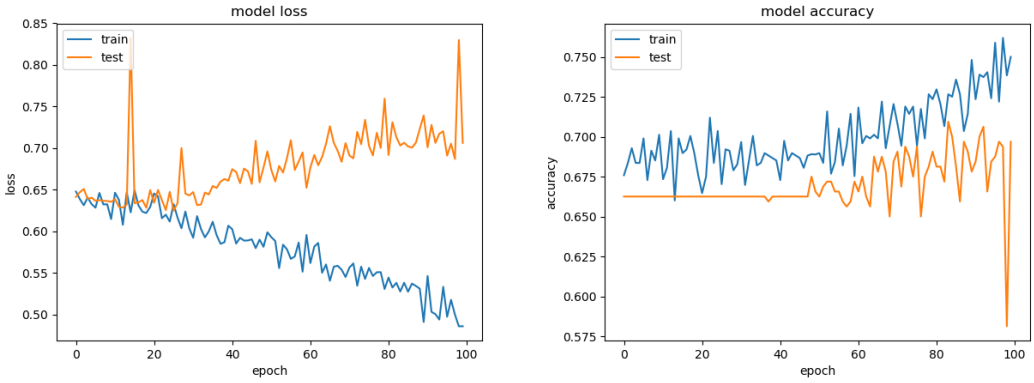
$$= -\frac{1}{N}\sum_i^N [y_i \log \hat{y}_i - (1-y_i)\log(1-\hat{y}_i)] \tag{5.7}$$

where $\hat{y}$ is the predicted value and y is the true value.

5.4.1 Training a small convolutional network from scratch

Our dataset consisted of images of the same size (300x300px), which we resized to 100x100px. We tested also different input image sizes, 150x150px and 50x50px. We used the same methods to randomly transform images as in the previous experiment, including rotation, shifting, shearing, zooming, and thus generating new data samples indefinitely.

Using the same small convolutional network as before, we achieved the average cross-validation accuracy of 66.54% (Table 5.4). This is even lower than the baseline 68%: accuracy of a naive classifier which always predicts the majority class. Progress of the loss function and accuracy during training is shown in Figure 5.17. It seems that the classifier is not able to learn general features to separate between the two classes.



(a) (b)

Figure 5.17: Loss and accuracy progress during training of the convolutional network for 100 epochs on the dataset of user ratings

Model	**Mean CV accuracy**	**CI 95%**
Baseline	*68%*	
Neural network	65.54%	56.78% – 74.29%
Random forest	66.84%	56.50% – 77.18%
Support vector machine	67.94%	59.89% – 75.99%
k-nearest neighbors	64.14%	55.25% – 73.03%

Table 5.5: Comparison of model accuracy of classifying images of the user ratings dataset using bottleneck representations from Inception v3 network

5.4.2 Retraining the Inception v3 network

Training a neural network on the extracted bottleneck features from the Inception v3 model, we achieved the average cross-validation accuracy of 65.54%. Several misclassified examples from a single CV fold are shown in Figure 5.18. Random forest and SVM on the bottleneck features did not fare better, they achieved 66.84% and 67.94% accuracy, respectively. Comparison of the results including 95% confidence intervals is shown in Table 5.5.

5.4.3 Retraining the last convolutional block and fully-connected layers of VGG16

When we trained a fully-connected neural network on the bottleneck features from VGG16, the average cross-validation accuracy was 66.94%. When in addition to that we fine-tuned also the last block of convolutional layers in the same way as in Section 5.3.3, we achieved the average CV accuracy of 67.73% (Table 5.6).

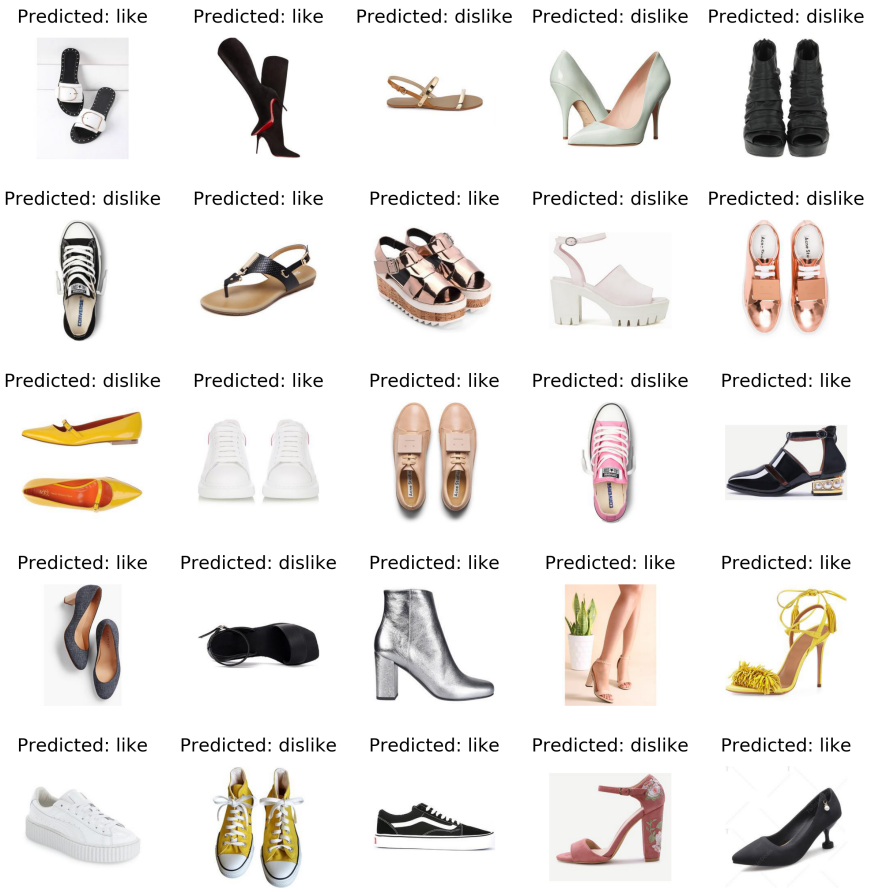


Figure 5.18: Incorrectly classified examples from the validation set

Model	**Mean CV accuracy**	**CI 95%**
Baseline	*68%*	
VGG16 top layers and conv block retraining	67.73%	61.58% – 73.88%

Table 5.6: Accuracy of classifying images of the user ratings dataset using VGG16 model with retrained last convolutional block and a fully-connected neural network

Model	Mean CV accuracy	CI 95%
Baseline	*11.1%*	
Small convolutional network	71.98%	66.15% – 77.81%
Inception v3 top layers retraining	78.59%	73.97% – 83.21%
VGG16 top layers and conv block retraining	94.94%	93.74% – 96.13%

Table 5.7: Comparison of model accuracy classifying images of the 9-shoe-types dataset

Model	Mean CV accuracy	CI 95%
Baseline	*68%*	
Small convolutional network	66.54%	61.46% – 71.62%
Inception v3 top layers retraining	65.54%	56.78% – 74.29%
VGG16 top layers and conv block retraining	67.73%	61.58% – 73.88%

Table 5.8: Comparison of model accuracy classifying images of the user shoe ratings dataset

5.5 Summary

Results of classification on the datasets of shoe types and user ratings using convolutional networks are summarized in Tables 5.7 and 5.8, respectively.

Although the task of classifying user's preferences may look similar to classifying types of clothing, even uses the same input data, the results suggest that it is more challenging. In the task of classifying types of shoes from 9 balanced classes, our best model achieved 95% accuracy. In classifying user's ratings, we have hardly exceeded baseline: the accuracy of a naive classifier which always predicts the majority class. The classes were slightly imbalanced, but weighting the classes proportionally to their size did not help much. Why is it possible to classify the images into categories by their type, but not by a user's taste?

The problem may be in the data itself. A person's preferences are not as obvious as it seems, even from a labeled dataset. They depend on minor details, cultural influences, and may even change during the process of rating. They may depend on features that can't be accurately determined from the limited amount of data. Another person, even an expert stylist, might not correctly recommend every piece of clothing upon seeing this dataset, while the previous task was easily solvable for an average person based on knowledge of a few features of the shoes. Also, we used only labeled data from a single person, and since various people have different preferences, the models could possibly achieve higher accuracy when using a different set of labels.

For future experiments, the models could make better decisions if they had more data available than only a single small set of labeled images. It might be useful to collect data about the preferences of multiple users, and to base the recommendations also on ratings from other users with a similar taste. Also, using a larger dataset of unlabeled images in addition to the one with the user's labels could help to extract the right features from the images.

6. Generating outfits

One of our main goals of the thesis was to explore the possibility of creating wearable outfits automatically. We will approach this by using the dataset of clothing that we have collected and attempt to combine them into outfits in a not completely random way. We will use supervised machine learning algorithms to recognize and distinguish good outfits from bad ones, in order to be able to optimize towards the better ones, and then use genetic algorithms to do this optimization, combining together clothing from our database.

Another option would be to learn a probability distribution from our dataset of outfits without the need of supervised labels. Such distribution would have to be created based on a set of features that represent an outfit. If we then generate an outfit from this distribution, we would have to transform it back into an actual set of images of clothing, that we could show to a user. We will try models such as restricted Boltzmann machine and deep belief network to approach the task in this way.

6.1 Data

In order to train a supervised machine learning algorithm what makes an outfit fashionable, we would like to have a labeled dataset with enough examples of good, as well as bad outfits. The task of collecting such dataset is made more difficult because the outfit's stylishness is subjective to each person, and depends on many external circumstances, including the time period, current trends, occasion, and so on.

6.1.1 Manually-created vs. random outfits

Our first approach was to use our dataset of outfits collected from Polyvore (all of which have received at least 100 likes) as the positive samples, and use randomly generated outfits as negative samples. The method of generating random outfits has been described in Chapter 3.2.5. This results in a balanced dataset with around 4000 examples, but more such data could be collected easily. Of course, these labels are not completely ground truth, as many random combinations of clothing may end up being quite good, and some of the manually created ones may not be to a particular person's taste. But they will show to be quite useful in distinguishing some elements of human-curated outfits, such as matching color schemes.

6.1.2 Users' ratings

Another approach to obtaining labeled data was to obtain positive and negative ratings of outfits. Using the web application from Chapter 3, we have collected volunteer ratings for both random and manually created outfits. Users rated each outfit between 1 (strongly dislike) and 5 (strongly like). From this, we tried averaging the users' ratings of an outfit to obtain a label, or using ratings

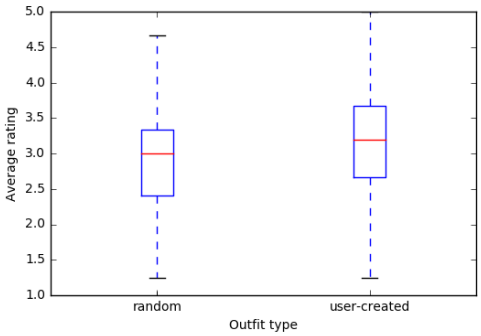


Figure 6.1: Distribution of the ratings according to an outfit's source: random or Polyvore. The average rating is 2.92 and 3.17, respectively.

from a single user to train a personalized recommendation model. There were 31 volunteers, each rated on average 368 outfits. However, with the relatively small number of data collected in this way, the averaged rating of volunteers did not turn out to be a very reliable measure, as the ratings of each outfit by multiple users often differed quite a lot.

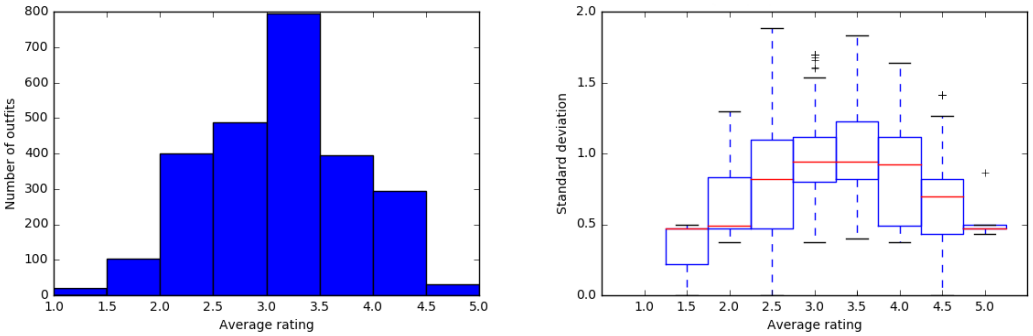
Of these ratings, 89% were submitted by female volunteers. The table below shows the distribution of age groups in the ratings, 42% were submitted by users in the age group 35-44. In Figure 6.1, we can see that there is not a large correlation between the user's rating and whether the outfit was generated randomly or obtained from Polyvore.

Age group	**Number of ratings**
35-44	4835
18-24	2194
25-34	1957
unknown	1246
45-54	600
under 18	540
55-64	46

As the dataset for training our models, we selected only outfits that had been rated by at least 3 users. From the original 4000 outfits, around 2500 fulfilled this condition. We can see the distribution of the averaged ratings, as well as the standard deviation within an outfit's ratings in each average rating range in Figure 6.2. The standard deviation is around 1 when the average rating is around 3.0. This shows that people differ quite a lot in their rating of many outfits, especially those that are not uniformly considered positive or negative. A small sample of the best- and worst-rated outfits is shown in figures 6.3 and 6.4.

6.1.3 Single user's ratings

In our data collection, we have encountered users that contributed with around 1500 outfit ratings. However, we cannot know for certain if it was a single person,



(a) Histogram of the average outfit ratings

(b) Distribution of the standard deviation within an outfit's ratings in each average rating range. The boxplots indicate its quartiles, with the mean value shown in red.

Figure 6.2: Distribution of the average rating and its standard deviation in outfits rated by at least 3 volunteers

Figure 6.3: Outfits with the highest average rating by volunteers

Figure 6.4: Outfits with the lowest average rating by volunteers

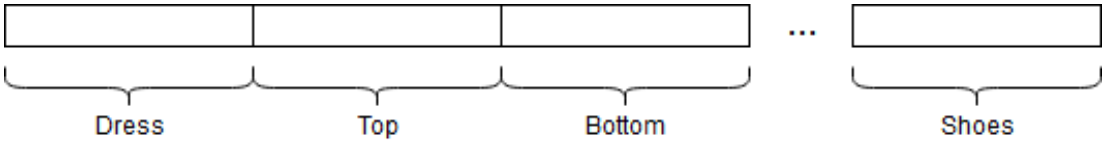


Figure 6.5: Outfit represented as a vector of features. Each clothing type corresponds to a fixed-length set of its features.

or multiple sharing the same account. We tried to also train a model based on only this user's ratings. The ratings were quite skewed toward the positive, with an average rating of 3.62.

6.2 Rating outfits

To train a machine learning model to rate outfits, we need to create a common representation for all outfits that we can feed as an input to the model. The outfits in our datasets all conform to the schema defined in Chapter 3.2.2, which ensures that there is at most one item of each type in the outfit. To simplify the problem, we will only use a subset of an outfit's possible clothing types, the main clothing without minor accessories, that is, *dress, top, bottom, outerwear, bag* and *shoes*. This is done so that we can focus on the most important relationships of pieces of clothing, while we test various models.

6.2.1 Fixed-length vector representation

To create a fixed-length vector representation which can be used by the standard machine learning models, we concatenate features representing each of the main clothing items in an outfit. Each item feature vector has a fixed length position and position in the resulting outfit vector (Fig. 6.5). The length of representation of the clothing types can differ, according to what features are needed to encode it.

Each item needs to be transformed to a set of appropriate features. In this experiment, we are using the item's colors from the image and its most specific category. The color scheme of the whole image (without the white background) has been encoded into a normalized color histogram with 64 bins, in a similar way as in Chapter 4.3.2. The category has been one-hot encoded into a vector with length equal to the number of possible categories which can belong to that clothing type. These two vectors are concatenated, forming the item's feature vector. The vectors of each item are then concatenated into the outfit's representation.

We have tested these input representations of our datasets with several machine learning models for classification (or regression when using real-valued labels), such as a fully-connected neural network with one hidden ReLU layer of 50 units and dropout, support vector machine and random forest.

The representation could be expanded also with textual data we have about the items, vectorizing titles and descriptions of the items after filtering only relevant keywords. Also, it would be useful to extract more interesting features from the images. It would also be possible to use the representation of the item's image in one of the large pretrained convolutional networks (e.g. Inception v3, VGG16), as in Section 5.2. However, it would create a very high-dimensional representation compared with the size of our dataset (the bottlenecks being vectors of length around 2000 for each image). In the preliminary experiments this approach did not prove very successful.

6.2.2 Image list representation

Since the previous representation does not contain all of the information from the image, another option is to use convolutional networks directly with the input images, to learn the appropriate features from the data. Since each sample in this dataset of outfits is a set of images that are labeled together, it is not as straightforward as using a single convolutional network. However, it is possible to concatenate the outputs of convolutional networks for each item's image in a similar way as static vectors, add several fully-connected layers on top of it, and propagate gradients in this joint network.

To allow for better generalization from our relatively small data set, as well as faster computation, we resized the input images to 50x50. We applied random image transformations, such as rotation, zoom, skew, to each item of an outfit in the training set.

The model that we used on these image sets, was created by concatenating outputs of convolutional networks (Fig. 6.6). In this joined network, each item type has a fixed position in the input, in order to learn type-specific image features and structure of an outfit. We kept the layers small, using three convolutional-maxpooling blocks of sizes 16, 24, and 32. The fully-connected layers had 32 neurons each, with ReLU activation function. The output unit had a sigmoid activation function, outputting the final score of the outfit between 0 and 1. Most of our problems involved binary classification, so we used the binary cross-entropy loss function, and Adam optimization algorithm [Kingma et al., 2014] for training the network. We added a small dropout in some cases when the network showed signs of overfitting.

It would be possible to further fine-tune many parameters of the network, such as the number of layers, sizes of the convolutional as well as fully-connected layers, activation functions, input size, regularization methods, and so on.

6.2.3 Results

We evaluated each of the two representations and models on our three labeled datasets. The models were evaluated using 10-fold cross-validation, reporting also 95% confidence intervals of the classification accuracy.

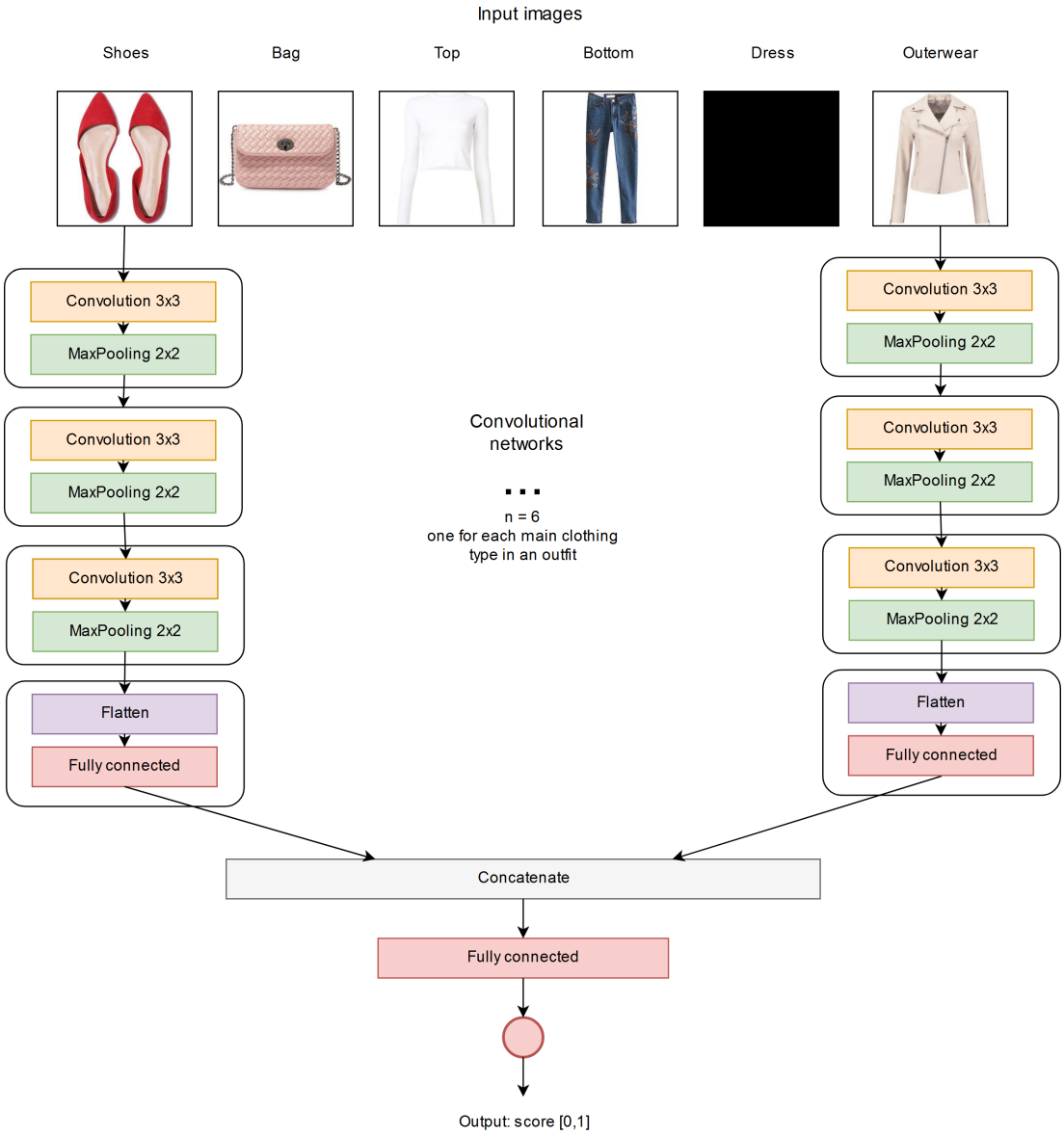


Figure 6.6: Model created by joining the outputs of several convolutional networks, with a fully-connected neural network on top of their joined output. The training is performed jointly by back-propagation. Each sub-network receives as its input the image of the predefined clothing type. A dress is not present in this outfit, therefore its inputs are zero.

Representation	Model	Mean CV accuracy	CI 95%
	Baseline	*55%*	
Vector of features	Neural network	64.62%	60.46% – 68.78%
Vector of features	Random forest	59.32%	56.08% – 62.55%
Vector of features	SVM	54.60%	49.33% – 59.87%
Images	Joined CNNs	63.26%	60.81% – 65.71%

Table 6.1: Comparison of model accuracy of classifying outfits from Dataset 1

Representation	Model	Mean CV accuracy	CI 95%
	Baseline	*55%*	
Vector of features	Neural network	60.05%	57.04% – 63.05%
Vector of features	Random forest	59.10%	51.75% – 66.46%
Vector of features	SVM	54.69%	47.19% – 62.20%
Images	Joined CNNs	58.63%	52.60% – 64.66%

Table 6.2: Comparison of model accuracy of classifying outfits from Dataset 2

Dataset 1: Polyvore vs. randomly generated

This was a binary classification task, where we had a dataset of 4000 outfits, 45% of which were from Polyvore (positive samples) and 55% were randomly generated (negative samples). Therefore, the baseline for this experiment (a model that always predicts that an outfit is negative), is 55%. The results for each of the proposed representations are shown in Table 6.1.

Dataset 2: Users' ratings

In this task of predicting the average user rating of an outfit, we first tested regression, trying to predict the average rating as a real number, but this did not yield very good results. The reported mean squared error of each tested model was similar to the variance in the labels, which would be the error of a baseline model that would always output the average label in the dataset.

We then turned this into a classification task, whether the average rating of an outfit was greater than 3. In this way, the classes were almost balanced, with 46% of outfits being in the positive class. The results of classification by our models are shown in Table 6.2. This dataset yielded the worst accuracy, which was expected, since the different users' ratings within an outfit varied a lot.

Dataset 3: Single user's ratings

In the task of classifying whether a user would like an outfit or not, we also used binary classification, whether the user's rating was greater than 3. There were 1500 samples in the dataset, 60% of which were in the positive class. The results are shown in Table 6.3.

Representation	Model	Mean CV accuracy	CI 95%
	Baseline	*60%*	
Vector of features	Neural network	63.37%	54.38% – 72.37%
Vector of features	Random forest	60.48%	55.94% – 65.03%
Vector of features	SVM	59.89%	50.85% – 68.94%
Images	Joined CNNs	63.38%	59.59% – 67.17%

Table 6.3: Comparison of model accuracy of classifying outfits from Dataset 3

Summary

On all three datasets, the basic neural network trained on the vector representation of each outfit and the joined convolutional networks both achieved the highest accuracy. The maximum accuracy was however only around 65%. It may have been caused by various reasons: quality of the labels, possibility that the classes are not completely distinguishable, small size of our dataset, not selecting the right parameters of our models, or just the inherent difficulty of the problem. We will show in the next section which outfits would these models score the highest.

These results might be improved by using a larger dataset of labeled data, which is feasible especially for Dataset 1: random vs. Polyvore. Further approach to improve the results might be to focus on better representation extraction in the case of using the vector of features. A way to deal with the small number of labeled data might be to use semi-supervised learning, where an algorithm would be trained on a larger unsupervised dataset of clothing images or outfits to extract useful features, but more specifically on a smaller number of data labeled by an expert.

6.3 Genetic algorithms

Having created a few models that are able to score an outfit (although not perfectly, judging by their accuracy), we can now use a genetic algorithm to generate new outfits that would be scored highly by these models, by using them as a fitness function for the GA. Also, genetic operators such as mutation and crossover might be suitable for this task, because changing and recombining pieces of clothing are a way in which humans approach styling as well.

For the implementation of genetic algorithms, we have used DEAP (Distributed Evolutionary Algorithms in Python) [Fortin et al., 2012] framework.

6.3.1 Representation

The representation of an outfit, or an individual in the GA, is a set of items, fulfilling the set of rules defined in Section 3.2.2. When generating or changing outfits, we only allow for operations that create outfits that conform to the rules. To simplify the task, we will be using only the main clothing types. In this

context, an outfit consists of a dress or a top and bottom, a bag, shoes, and optionally outerwear, but this could be extended to other accessories.

6.3.2 Genetic operators

Mutation

Mutation of an outfit is performed as follows: for each type of clothing present in an outfit, with probability p we replace it with a random item of the same type from our database. We set the probability to 0.25, in order to perform on average one mutation per outfit. For optional items, we use their prior probability to add or remove them from the outfit. Also, a dress can be randomly swapped for a top and bottom, and vice versa.

Another option would be to select a new item with the probability corresponding to its distance from the current item in some feature space (such as the vector of features proposed in Section 6.2.1, or an encoding obtained from an image).

Crossover

We use a variation of uniform crossover customized for this representation. For each pair of items of the same type occurring in the two outfits, we swap them with probability p, the probability being again 0.25. We need to be careful when crossing an outfit which has a top and bottom with another which has a dress so we do not end up with a combination that is not allowed by our schema.

6.3.3 Fitness function

As a fitness function of the GA, we need a method to evaluate whether an outfit is good, or alternatively, whether a particular user is going to like it. We use a trained machine learning model, which outputs a value $y = f(\mathbf{x})$, where $\mathbf{x}$ is a vector of features representing the outfit and $y \in \langle 0, 1 \rangle$ is the predicted score of an outfit. We will test two of our models from the previous section, trained on the dataset of random vs. Polyvore outfits:

1. Multi-layer neural network which takes as its input a vector of colors and categories of each clothing item in the outfit. This network achieved 63% cross-validation accuracy.

2. Joined convolutional neural networks taking as the input images of clothing items in an outfit, with 65% cross-validation accuracy.

6.3.4 Results

After running a simple genetic algorithm, we first find that many of the best outfits repeat in the population. It is quite easy for the GA to find a solution with the fitness close to 1. This is not our only objective, and it is further complicated by the fact that our neural network is not an absolute measure of truth. We would like to generate a more diverse set of outfits to show to a user, and also in this way illustrate what the neural network considers as the best outfits. We can

also compare the resulting outfits to those created randomly (of which a sample was shown in Figure 3.3), which can serve as a kind of visual baseline for this task.

The outfits generated in one population end up being very similar, even after removing exact duplicates. Once the algorithm finds a solution that has fitness close to one, it prevails in the population. Therefore, to obtaining varied results, we run the genetic algorithm multiple times, and select a few of the best items in each. We run the GA 4 times with independent populations, and select 3 best individuals from each resulting population, to illustrate the small variations in the outfits that the model considers to be the best. The results are shown in Figure 6.7 when using the simple neural network to evaluate fitness, and Figure 6.8 using the joined convolutional network.

Each run provides a different sample of outfits. We can only hypothetise whether the genetic algorithm generates actually good outfits. Either way, it seems to pick outfits with some purpose. The challenge was that the models that we used as a measure of the outfit's score did not have a high accuracy. They were trained on data with not completely reliable labels, and the training dataset was quite small. However, the outfits with the highest score by our models are still most likely better than random in this highly subjective task.

In the case of the outfit representation made up of colors and categories (Fig. 6.7), the resulting outfits have quite bold colors, often with clashing combination. But the colors in most of these outfits match quite well. When there is a clash, for example of pink and red, it is because in this representation, these colors end up in the same bin, and for the model are therefore indistinguishable. In the case of joined convolutional networks (Fig. 6.8), the outfits seem much more wearable. We can also see that when swapping items in an outfit for other, visually similar items, the resulting score of the outfit does not change much, which is an expected behavior.

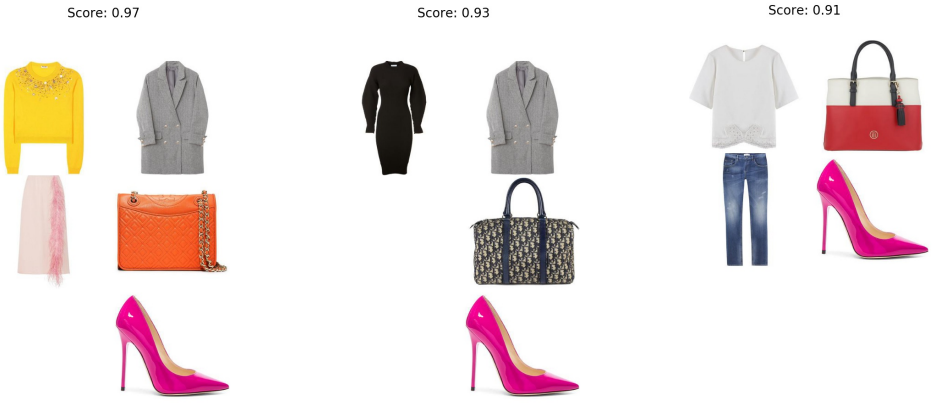
It is interesting to look at each batch of these outfits, and see that some of them are very wearable, while others are quite wild. But in between, new and inspiring clothing combinations are already starting to emerge, where we can imagine a person wearing them. As a bottom line, one could always argue that a particular outfit is not ugly – it is just intentional in line with the 'ugly fashion' trend [Stoppard, 2017]!

6.3.5 Selection from a user's wardrobe

The same genetic algorithm can be also used to create combinations from a smaller clothing set. As mentioned in the beginning the thesis, we would like to be able to generate outfits from a user's wardrobe, in order to help the user better utilize the clothing she owns. We have created a test wardrobe of a user (Figure 6.9), on which we tried our genetic algorithm, whether it will be able to generate wearable outfits using only this set of 36 items.

(a) Top 3 outfits from run 1

(b) Top 3 outfits from run 2



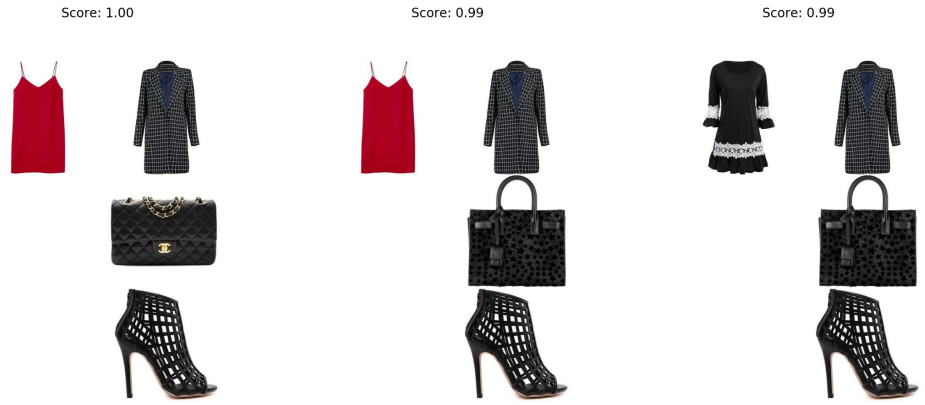
(c) Top 3 outfits from run 3

(d) Top 3 outfits from run 4

Figure 6.7: Best outfits generated by the GA, scored by a neural network with the representation using colors and category of each clothing item

(a) Top 3 outfits from run 1

(b) Top 3 outfits from run 2



(c) Top 3 outfits from run 3

(d) Top 3 outfits from run 4

Figure 6.8: Best outfits generated by the GA, scored by the joined CNN on images of clothing items

Figure 6.9: Test wardrobe of a user

The results are shown in Figures 6.10 and 6.11, for the neural network and joined convolutional networks, respectively. With a small, manually selected wardrobe of items, the results look even better than before. It is more feasible to generate outfits tailored to a style of a user if we use only the items that she likes or owns.

Similarly, if we wanted to recommend to a user some options of what to wear with a particular clothing item, it would be possible to fix the item in place during the genetic operations in order to generate only outfits that include that specific item.

6.4 Deep belief networks

Another approach to generating outfits is to model a probability distribution that we can obtain from our dataset of outfits from Polyvore, manually created by stylists. From this probability distribution, we would like to be able to generate a new outfit. Generative unsupervised models such as restricted Boltzmann machines and deep belief networks can be useful for this task. We will test and compare the results using the two models.

6.4.1 Generating outfits

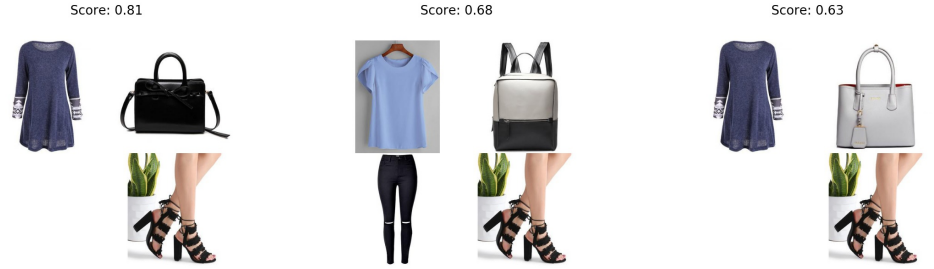
First, we need to represent each outfit as a fixed-length vector, so that any outfit can be encoded and then reconstructed as a set of images that can be shown to a user. As in Section 6.2.1, we will use vector representation of each of its items, consisting of normalized three-dimensional color histogram of each item's image and the item's category in one-hot encoding. We will be using only the main clothing types from the outfits (*dress, top, bottom, shoes, bag, outerwear*), and concatenate their representations. The resulting vectors have values between $\langle 0, 1 \rangle$.

When transforming the vector representation back to a set of images, we can retrieve the closest item of each type in our dataset, by finding one from the category that has the highest activation, and whose color scheme is the most similar, by means of Euclidean distance between their vectors. Since the resulting color vectors of each item output from a RBM or DBN may have different magnitudes, we scaled each to sum up to one. However, encoding an outfit in this way, as well as the retrieval approach, introduces some loss of information.

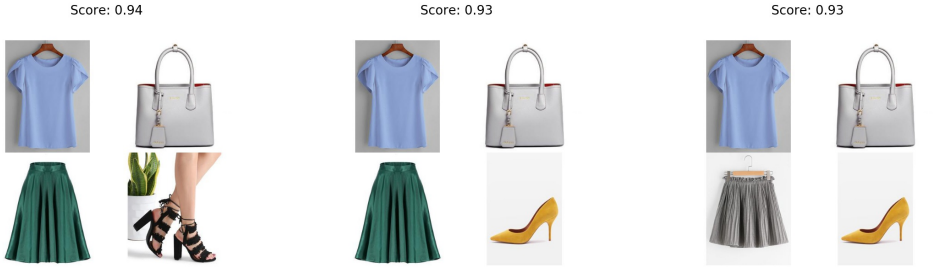
We trained each of the models on the dataset of 1800 outfits obtained from Polyvore. The length of each outfit vector encoded in this way was 442 (each of the 6 main clothing types was represented as a color histogram with 64 bins + several options for the category).

To generate new outfits, we need to also provide some input to the network, so that we can run the forward and backward computations until it converges to a stable state. First, we reconstructed several outfits from the training set in

(a) Top 3 outfits from run 1



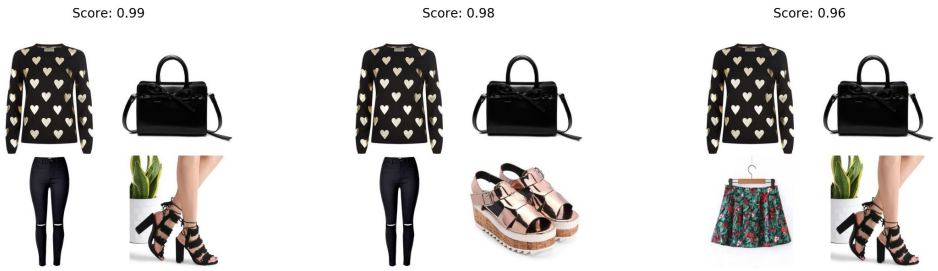
(b) Top 3 outfits from run 2



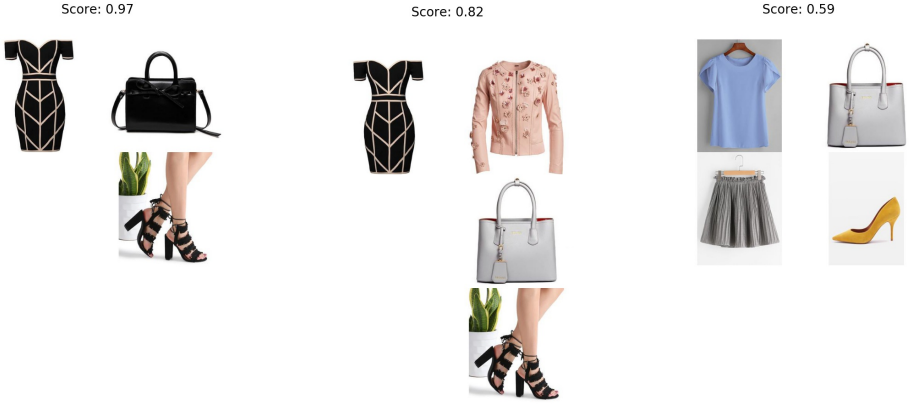
(c) Top 3 outfits from run 3

(d) Top 3 outfits from run 4

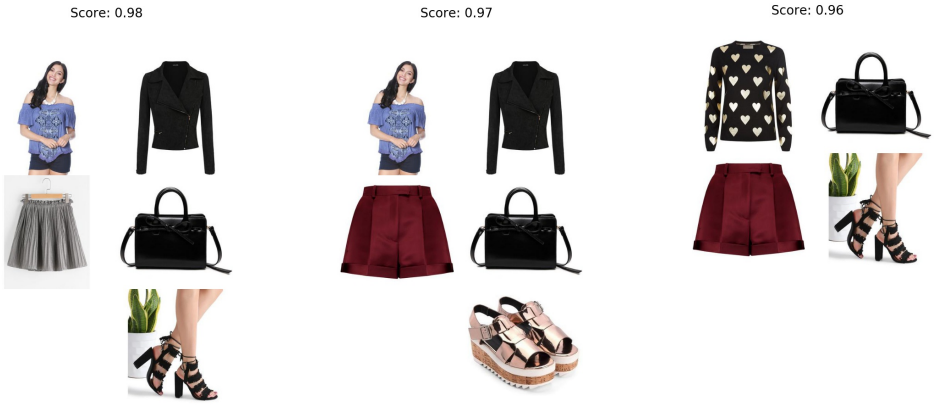
Figure 6.10: Best outfits generated from items in the test user's wardrobe. Scored by the neural network using vector outfit representation



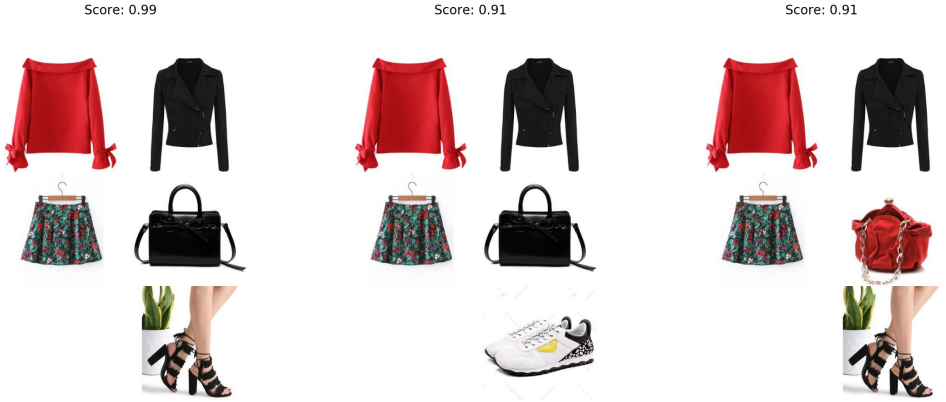
(a) Top 3 outfits from run 1



(b) Top 3 outfits from run 2



(c) Top 3 outfits from run 3



(d) Top 3 outfits from run 4

Figure 6.11: Best outfits generated from items in the test user's wardrobe. Scored by the joined CNNs

this way to see if the models result in a similar outfit. However, we would like to be able to generate completely new outfits. We first tried generating from a random noise, but that did not yield good results. Then, we initialized the RBM or DBN to a randomly generated outfit and ran the reconstruction for several epochs until it reached a stable state. This resulted in outfits slightly similar to the input outfits, but (presumably) with some information they have learned from the training dataset.

The training required a lot of parameter tuning, which for both models included the hidden layer sizes, number of epochs, learning rate, and batch size. The goal was to achieve a low average reconstruction error (distance) between the training set input and the reconstructed output. The resulting outfits mostly had the same structure as the input ones, often with the same categories of clothing.

6.4.2 Results

The model of a binary RBM that yielded the best performance had 400 units in the hidden layer. We trained it for 50 epochs using mini-batches of 10 samples each and learning rate 0.1. After the unsupervised training on our dataset of outfits, we reached the average reconstruction error of 0.52. Reconstructed outfits from a sample of training set outfits are shown in Figure 6.12, and are compared with the outputs of a DBN. We can see that the reconstruction resulted in outfits with items of similar categories, but often having different colors. It was less difficult for the network to reconstruct a similar outfit from a common outfit, such as in Figures 6.12a and 6.12d, than from a unique one. Reconstructed outfits from a sample of randomly generated outfits are shown in Figure 6.13. We also evaluated the original and reconstructed outfits using the convolutional network from 6.2.2, trained to classify random vs. Polyvore outfits.

Next, we used a DBN with two layers of binary RBMs, 400 units each, which we trained in an unsupervised way. We used learning rate of 0.1, batch sizes of 20, and trained each RBM for 30 epochs. We reached the reconstruction error of 0.76 in the first hidden layer and 1.59 in the second hidden layer. The reconstruction error of the second layer was always higher, because the representations in the first layer are probably more difficult to model than the inputs. The comparison with RBM in reconstructing outfits from a sample of training set outfits is shown in Figure 6.12. Since the error was higher in the case of DBN, it resulted in outfits that differed more from the training set inputs. Reconstructed outfits from a sample of randomly generated outfits are shown in Figure 6.14, also evaluated by a network trained to classify outfits. We can see that some of the items occur repeatedly in the small sample of reconstructed outfits. These might correspond to stable states of the network.

6.4.3 Including a user's rating

We also tested an approach to generating outfits that we would recommend to a particular user. We used data on outfit rating from one of our volunteers. Of the 1800 outfits in the dataset from Polyvore, 258 were rated as negative by

(a) Original outfit (left), reconstruction using RBM (middle) and DBN (right)

(b) Original outfit (left), reconstruction using RBM (middle) and DBN (right)

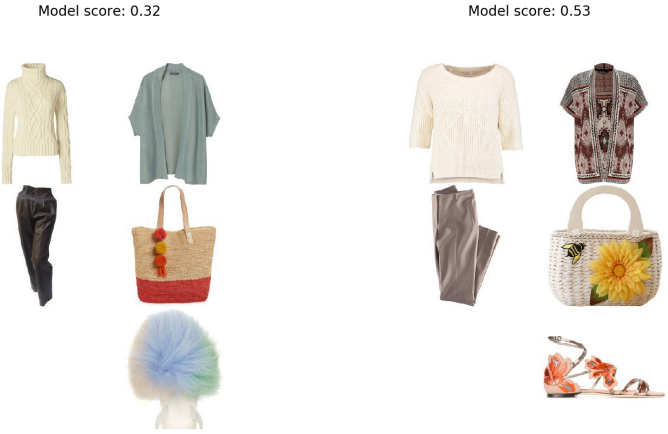
(c) Original outfit (left), reconstruction using RBM (middle) and DBN (right)

(d) Original outfit (left), reconstruction using RBM (middle) and DBN (right)

Figure 6.12: Comparison of reconstructed outfits by the RBM and DBN from the training set outfits

(a) Original (left) and reconstructed (right) outfit

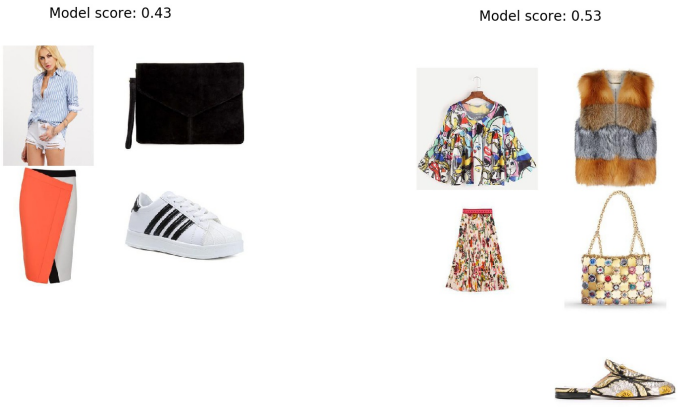
(b) Original (left) and reconstructed (right) outfit



(c) Original (left) and reconstructed (right) outfit

(d) Original (left) and reconstructed (right) outfit

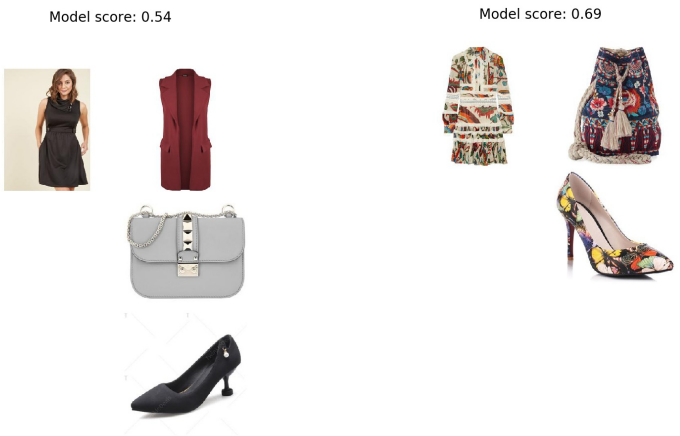
Figure 6.13: Reconstructed outfits by the RBM from randomly generated outfits. Scores were evaluated by the joined convolutional networks



(a) Original (left) and reconstructed (right) outfit

(b) Original (left) and reconstructed (right) outfit

(c) Original (left) and reconstructed (right) outfit



(d) Original (left) and reconstructed (right) outfit

Figure 6.14: Reconstructed outfits by the DBN from randomly generated outfits. Scores were evaluated by the joined convolutional networks

the user, and 426 as positive. We included two more units to the input with our original outfit vectors in an attempt to learn the joint probability distribution between an outfit and whether the user liked it. These two inputs corresponded to like and dislike, and would be set to either 1 or 0. Since there were also outfits that were not rated by the user, we included them in training as well.

In this way we tried to associate features of the outfits with whether the user liked or disliked the outfit. When generating an outfit that the user would like, we would supply a randomly generated outfit as an input and set the input unit corresponding to 'like' to 1, and the other one to 0. This usually resulted in outputs that has the same values of these two units. It is possible that the reconstruction was performed in the same way as before, without much regard to the user rating unit.

We used the same model parameters as in the previous section. The binary RBM resulted in the reconstruction error of 0.52. A sample of reconstructed outfits, to which the RBM predicted positive rating by the user, are shown in Figure 6.15. The DBN resulted in reconstruction error of 0.76 in the first layer, and 1.59 in the second layer. A sample of reconstructed positive outfits by the DBN are shown in Figure 6.16.

6.5 Summary

In this chapter, we proposed two ways to represent an outfit so that it can be classified by a machine learning model. First, we used a vector of features based on colors and categories of each clothing type in the outfit. We trained a neural network to classify whether an outfit is good or not, for example by using a dataset of random outfits as negative samples, and outfits from Polyvore as positive samples. Second, we proposed and implemented a model of joined convolutional networks which extracts features directly from each of the images of clothing items that are part of the outfit, and trained it using the same labels.

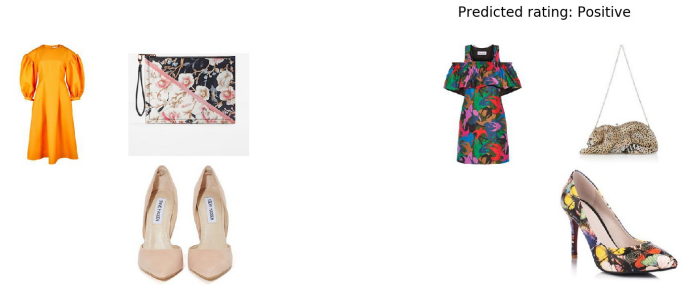
Based on these representations, we tested several approaches to generating outfits, each producing different kinds of outfits. This task is difficult to evaluate objectively, but we would consider the genetic algorithm with the joined convolutional neural networks as its fitness function to generate the most wearable outfits, even though the convolutional network itself did not reach very high accuracy in classifying random vs. manually created outfits.

In case of the generative models such as RBM and DBN, it is difficult to evaluate whether they actually learned to generate good outfits, apart from the reconstruction error. It is an approach that results in a larger loss of information than the genetic algorithm, which is introduced by encoding the outfits to a simplified representation, and then reconstructing them by finding the most similar item from our dataset. Nevertheless, it created a few interesting outfits, without the need of using an auxiliary dataset of random outfits as the negative-labeled set.

(a) Original (left) and reconstructed (right) outfit

(b) Original (left) and reconstructed (right) outfit

(c) Original (left) and reconstructed (right) outfit



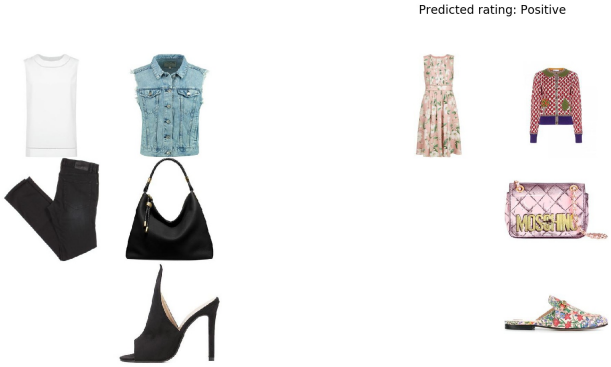
(d) Original (left) and reconstructed (right) outfit

Figure 6.15: Reconstructed outfits by the RBM from randomly generated outfits, with predicted positive rating by the user

(a) Original (left) and reconstructed (right) outfit

(b) Original (left) and reconstructed (right) outfit

(c) Original (left) and reconstructed (right) outfit



(d) Original (left) and reconstructed (right) outfit

Figure 6.16: Reconstructed outfits by the DBN from randomly generated outfits, with predicted positive rating by the user

The outfit rating accuracy could be further improved by using larger datasets, and extracting more relevant features from the clothing items, for example through the use of pre-trained models. Generating outfits could be improved by using more advanced rules than our simple schema allows. Generating personalized recommendations of outfits would be more effective if we combined subjective data provided by a user with a larger number of examples having reliable and more objective labels.

Conclusion

This thesis is devoted to deep neural networks and their applications in fashion. Our goal was to design a viable strategy for generating attractive fashion outfits based on user-defined preferences. In our case, these preferences were specified in the form of labeled data-sets provided by real people. For this (rather general) scenario, we have developed and implemented several generative models and methodologies. The thesis presented a series of experiments on a novel dataset of clothing items and outfits composed from them by stylists. In order to obtain enough data for training, we created a prototype of a web application, and collected data from volunteers about their fashion preferences. Within the framework of this thesis, we then succeeded in introducing a proof of concept of several clothing and outfit recommendation methods.

Partial solutions to the main task include clustering of colors present in the images of the considered clothing items in order to find dominant colors, which could provide information about the piece of clothing. Convolutional neural networks were used to extract further useful representations for the processed data. In this context, we used pre-trained models such as Inception v3 and VGG16 to explore and visualize the computed internal representations. Within the framework of transfer learning, pre-trained network models have been re-used and re-trained on our fashion data in order to improve their classification performance. The methods explored to generate new outfits include genetic algorithms, convolutional neural networks and deep belief networks.

Since this topic deals with data that is by nature subjective, it makes evaluation of results difficult. Unfortunately, we cannot assign a ground truth label to each outfit, as one person may have a different opinion than another. For this reason, we tried also crowd-sourcing ratings of outfits, but these varied a lot within the small data sample. We also tried targeting this outfit recommendation to a specific user.

In classification of a small dataset of fine-grained clothing categories, such as types of shoes, we achieved 95% cross-validation accuracy using a pre-trained VGG16 network, in which we re-trained a fully connected classifier, as well as the last block of convolutional layers. Using the same methods to classify the same images according to a user's rating, which was either 'like' or 'dislike', proved much more difficult. In this task, we have hardly exceeded baseline accuracy. These classes may be harder to distinguish, because the user's opinion may depend on smaller details not easily captured from the small dataset. Instead of evaluating clothing items alone, it might be better to look at how they can be combined with other pieces of clothing into outfits. In this way, the method of generating outfits could prove useful also for recommending specific pieces of clothing to a user.

We kept our dataset sizes purposefully small, because these preliminary experiments were done only on a single computer with a GPU. Using more pow-

erful computational resources would allow for experimentation with much larger datasets, which could lead to better results.

The main result of the thesis consists in the development of novel methods for outfit generation. We designed a schema of clothing types in an outfit that helps us to restrict generated outfits to plausible looking ones. As one of the options, we used genetic algorithms to combine pieces of clothing, where the fitness function of an outfit was evaluated by a model trained to distinguish between random outfits and outfits manually created by stylists. One of the models that we used were joined convolutional neural networks, one for each input image corresponding to a clothing type. In this way, the genetic algorithm was able to generate many stylish and wearable outfits. However, not all outfits generated by the algorithm were good, and they would need to be reviewed by a human stylist.

The other approach we introduced for generating outfits uses restricted Boltzmann machines and deep belief networks, which were applied to learn the distribution in our dataset of curated outfits. We used a simplified representation of an outfit, which consisted only of colors and categories of each item. This results in a loss of information, both in encoding the outfit, and decoding it by finding items that are most similar items to the representation vectors.

Future work

There are several ways in which we could improve the performance in recommending clothing items and generating outfits. Probably the most important improvement would be achieved by obtaining a larger amount of quality data. For example, outfit generating might be improved by obtaining a reliable labeled dataset of outfits. Adding more data would require more computational resources, but there is still room to grow by using available computing infrastructures. Since the recommendations of outfits and clothing change in time, this dimension would need to be incorporated as well, for example, by retraining the algorithms on new data, which would include new clothing and fashion trends. Our schema does not permit creating every type of outfit, and allows only one item of each clothing type, so it could also be extended.

The performance of models for clothing recommendation could be improved by extracting more relevant representations, before using supervised classification. This could be done for example by using a Siamese network to encode multiple representations of the same item (image and text, or different images), targeting them to be encoded to similar representations. When deciding whether to recommend a clothing item to a user, we could include the information about the preferences of other users with similar personal style. Furthermore, it would be useful to look at what outfits could be composed with the clothing item, which is where our methods of generating outfits could be helpful.

We used models pre-trained on general datasets. If similar models were trained on a large fashion-specific dataset with detailed classes, we could extract more

useful features for the distributed vector representation of the clothing items. Additional information would come from using text data, which we did not use in these experiments. It could be encoded into a distributed representation, for example, by using some of the word-to-vector embeddings.

All of these models could be further incorporated into a larger system with a user interface, such as the web application that we designed. This application could then be released to users, to obtain more data for training our models, which could in turn help in recommendation.

Bibliography

ABADI, Martín et al., 2015. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Available also from: `https://www.tensorflow.org/`. Software available from tensorflow.org.

BELL, Sean; BALA, Kavita, 2015. Learning Visual Similarity for Product Design with Convolutional Neural Networks. *ACM Trans. Graph.* Vol. 34, no. 4, pp. 98:1–98:10. ISSN 0730-0301. Available from DOI: `10.1145/2766959`.

BOSSARD, Lukas; DANTONE, Matthias; LEISTNER, Christian; WENGERT, Christian; QUACK, Till; VAN GOOL, Luc, 2013. Apparel Classification with Style. In: *Apparel Classification with Style. Proceedings of the 11th Asian Conference on Computer Vision - Volume Part IV*. Daejeon, Korea: Springer-Verlag, pp. 321–335. ACCV'12. ISBN 978-3-642-37446-3. Available from DOI: `10.1007/978-3-642-37447-0_25`.

BRACHER, Christian; HEINZ, Sebastian; VOLLGRAF, Roland, 2016. Fashion DNA: Merging Content and Sales Data for Recommendation and Article Mapping. *CoRR*. Vol. abs/1609.02489. Available also from: `http://arxiv.org/abs/1609.02489`.

CHOLLET, François et al., 2015. *Keras* [`https://keras.io`].

CHOLLET, Francois, 2016. *Building powerful image classification models using very little data*. Available also from: `https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html`. Accessed: 2017-07-13.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L., 2009. ImageNet: A Large-Scale Hierarchical Image Database. In: *ImageNet: A Large-Scale Hierarchical Image Database. CVPR09*.

DI, W.; WAH, C.; BHARDWAJ, A.; PIRAMUTHU, R.; SUNDARESAN, N., 2013. Style Finder: Fine-Grained Clothing Style Detection and Retrieval. In: *Style Finder: Fine-Grained Clothing Style Detection and Retrieval. 2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 8–13. ISSN 2160-7508. Available from DOI: `10.1109/CVPRW.2013.6`.

DJANGO SOFTWARE FOUNDATION, 2017. *Django*. Version 1.10.5. Available also from: `https://djangoproject.com`.

DUCHI, John; HAZAN, Elad; SINGER, Yoram, 2010. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. Available also from: `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.html`. Technical report. EECS Department, University of California, Berkeley.

FORTIN, Félix-Antoine; DE RAINVILLE, François-Michel; GARDNER, Marc-André; PARIZEAU, Marc; GAGNÉ, Christian, 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research*. Vol. 13, pp. 2171–2175.

GEMAN, Stuart; GEMAN, Donald, 1984. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Trans. Pattern Anal. Mach. Intell.* Vol. 6, no. 6, pp. 721–741. ISSN 0162-8828. Available from DOI: 10.1109/TPAMI.1984.4767596.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron, 2016. *Deep Learning*. ISBN 978-0131471399. Available also from: http://www.deeplearningbook.org. Book in preparation for MIT Press.

HAN, Xintong; WU, Zuxuan; JIANG, Yu-Gang; DAVIS, Larry S., 2017. Learning Fashion Compatibility with Bidirectional LSTMs. *CoRR*. Vol. abs/1707.05691. Available from arXiv: 1707.05691.

HARA, Kota; JAGADEESH, Vignesh; PIRAMUTHU, Robinson, 2016. Fashion apparel detection: The role of deep convolutional neural network and pose-dependent priors. In: *Fashion apparel detection: The role of deep convolutional neural network and pose-dependent priors. IEEE Winter Conference on Applications of Computer Vision (WACV)*.

HASTIE, Trevor J.; TIBSHIRANI, Robert John; FRIEDMAN, Jerome H., 2009. *The elements of statistical learning : data mining, inference, and prediction*. New York: Springer. Springer series in statistics. ISBN 978-0-387-84857-0. Available also from: http://opac.inria.fr/record=b1127878. Autres impressions : 2011 (corr.), 2013 (7e corr.)

HAYKIN, Simon, 2008. *Neural Networks and Learning Machines*. 3rd ed. Pearson.

HINTON, Geoffrey E.; OSINDERO, Simon; TEH, Yee-Whye, 2006. A Fast Learning Algorithm for Deep Belief Nets. *Neural Comput.* Vol. 18, no. 7, pp. 1527–1554. ISSN 0899-7667. Available from DOI: 10.1162/neco.2006.18.7.1527.

HSIAO, Wei-Lin; GRAUMAN, Kristen, 2017. Creating Capsule Wardrobes from Fashion Images. *CoRR*. Vol. abs/1712.02662. Available from arXiv: 1712.02662.

JING, Yushi; LIU, David; KISLYUK, Dmitry; ZHAI, Andrew; XU, Jiajing; DONAHUE, Jeff; TAVEL, Sarah, 2015. Visual Search at Pinterest. *CoRR*. Vol. abs/1505.07647. Available also from: http://arxiv.org/abs/1505.07647.

JOHNSON, Emma, 2015. *The Real Cost of Your Shopping Habits*. Available also from: http://www.forbes.com/sites/emmajohnson/2015/01/15/the-real-cost-of-your-shopping-habits/#1158361d21ae. Accessed: 2017-27-01.

JOSHI, Nirav, 2016. *Elite Admin - Multipurpose Bootstrap 4 Admin Template*. Available also from: https://themeforest.net/item/elite-admin-the-ultimate-dashboard-web-app-kit-material-design/16750820.

KALANTIDIS, Yannis; KENNEDY, Lyndon; LI, Li-Jia, 2013. Getting the Look: Clothing Recognition and Segmentation for Automatic Product Suggestions in Everyday Photos. In: *Getting the Look: Clothing Recognition and Segmentation for Automatic Product Suggestions in Everyday Photos. Proceedings of the 3rd ACM Conference on International Conference on Multimedia Retrieval*. Dallas, Texas, USA: ACM, pp. 105–112. ICMR '13. ISBN 978-1-4503-2033-7. Available from DOI: 10.1145/2461466.2461485.

KANG, Wang-Cheng; FANG, Chen; WANG, Zhaowen; MCAULEY, Julian, 2017. Visually-Aware Fashion Recommendation and Design with Generative Image Models.

KIAPOUR, M. H.; HAN, X.; LAZEBNIK, S.; BERG, A. C.; BERG, T. L., 2015. Where to Buy It: Matching Street Clothing Photos in Online Shops. In: *Where to Buy It: Matching Street Clothing Photos in Online Shops. 2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 3343–3351. Available from DOI: 10.1109/ICCV.2015.382.

KINGMA, Diederik P.; BA, Jimmy, 2014. Adam: A Method for Stochastic Optimization. *CoRR*. Vol. abs/1412.6980. Available from arXiv: 1412.6980.

KULLBACK, S.; LEIBLER, R. A., 1951. On Information and Sufficiency. *Ann. Math. Statist.* Vol. 22, no. 1, pp. 79–86. Available from DOI: 10.1214/aoms/1177729694.

LECUN, Y.; BENGIO, Y.; HINTON, G., 2015. Deep learning. *Nature*. No. 521, pp. 436–444.

LI, Yuncheng; CAO, LiangLiang; ZHU, Jiang; LUO, Jiebo, 2016. Mining Fashion Outfit Composition Using An End-to-End Deep Learning Approach on Set Data. Vol. PP.

LIU, Si; FENG, Jiashi; SONG, Zheng; ZHANG, Tianzhu; LU, Hanqing; XU, Changsheng; YAN, Shuicheng, 2012. Hi, Magic Closet, Tell Me What to Wear! In: *Hi, Magic Closet, Tell Me What to Wear! Proceedings of the 20th ACM International Conference on Multimedia*. Nara, Japan: ACM, pp. 619–628. MM '12. ISBN 978-1-4503-1089-5. Available from DOI: 10.1145/2393347.2393433.

LIU, Ziwei; LUO, Ping; QIU, Shi; WANG, Xiaogang; TANG, Xiaoou, 2016. DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. In: *DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

MAATEN, Laurens van der; HINTON, Geoffrey E., 2008. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research*. Vol. 9, pp. 2579–2605.

MITCHELL, Tom M., 1997. *Machine Learning*. WCB McGraw-Hill.

OXFAM, 2016. *3.6 billion clothes left unworn in the nation's wardrobes, survey finds*. Available also from: http://www.oxfam.org.uk/media-centre/press-releases/2016/06/over-three-billion-clothes-left-unworn-in-the-nations-wardrobes-survey-finds. Accessed: 2017-27-01.

POLYVORE, 2017. *Polyvore: Discover and Shop the Latest in Fashion, Beauty and Home*. Available also from: http://www.polyvore.com/. Accessed: 01-04-2017. The website is no longer active because of its acquisition by SSENSE and the original content has been removed.

ROUSSEEUW, Peter J., 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*. Vol. 20, pp. 53 –65. ISSN 0377-0427. Available from DOI: http://dx.doi.org/10.1016/0377-0427(87)90125-7.

SIMO-SERRA, E.; FIDLER, S.; MORENO-NOGUER, F.; URTASUN, R., 2015. Neuroaesthetics in fashion: Modeling the perception of fashionability. In: *Neuroaesthetics in fashion: Modeling the perception of fashionability. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 869–877. ISSN 1063-6919. Available from DOI: 10.1109/CVPR.2015.7298688.

SIMONYAN, Karen; ZISSERMAN, Andrew, 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR*. Vol. abs/1409.1556. Available also from: http://arxiv.org/abs/1409.1556.

SMOLENSKY, Paul, 1986. Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory. In: RUMELHART, David E.; MCLELLAND, James L. (eds.). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*. MIT Press, 194–281.

SONG, Xuemeng; FENG, Fuli; LIU, Jinhuan; LI, Zekun; NIE, Liqiang; MA, Jun, 2017. NeuroStylist: Neural Compatibility Modeling for Clothing Matching. In: *NeuroStylist: Neural Compatibility Modeling for Clothing Matching. Proceedings of the 2017 ACM on Multimedia Conference*. Mountain View, California, USA: ACM, pp. 753–761. MM '17. ISBN 978-1-4503-4906-2. Available from DOI: 10.1145/3123266.3123314.

STATISTA, 2017. *Facts on the Apparel market in the U.S.* Available also from: https://www.statista.com/topics/965/apparel-market-in-the-us/. Accessed: 2017-27-01.

STOPPARD, Lou, 2017. *Why is fashion so ugly?* Available also from: https://www.ft.com/content/3d5ad52c-9f01-11e7-8b50-0b9f565a23e1. Accessed: 2018-03-22.

SZEGEDY, Christian; VANHOUCKE, Vincent; IOFFE, Sergey; SHLENS, Jonathon; WOJNA, Zbigniew, 2015. Rethinking the Inception Architecture for Computer Vision. *CoRR*. Vol. abs/1512.00567. Available also from: http://arxiv.org/abs/1512.00567.

TANGSENG, P.; YAMAGUCHI, K.; OKATANI, T., 2017. Recommending Outfits from Personal Closet. In: *Recommending Outfits from Personal Closet. 2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 2275–2279. Available from DOI: 10.1109/ICCVW.2017.267.

TENSORFLOW. *Inception in TensorFlow*. Available also from: https://github.com/tensorflow/models/tree/master/research/inception.

YAMAGUCHI, Kota; OKATANI, Takayuki; SUDO, Kyoko; MURASAKI, Kazuhiko; TANIGUCHI, Yukinobu, 2015. Mix and Match: Joint Model for Clothing and Attribute Recognition. In: XIE, Xianghua; JONES, Mark W.; TAM, Gary K. L. (eds.). *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, pp. 51.1–51.12. ISBN 1-901725-53-7. Available from DOI: 10.5244/C.29.51.

ZOGHBI, Susana; HEYMAN, Geert; GOMEZ, Juan Carlos; MOENS, Marie-Francine, 2016. Fashion Meets Computer Vision and NLP at e-Commerce Search. *International Journal of Computer and Electrical Engineering vol. 8*. Vol. 8, no. 1, pp. 31–43.

List of Figures

List of Tables

List of Abbreviations

CNN Convolutional neural network

CV Cross-validation

DBN Deep belief network

GA Genetic algorithm

LSTM Long short-term memory

RBM Restricted Boltzmann machine

ReLU Rectified linear unit

SVM Support vector machine

t-SNE t-Distributed stochastic neighbor embedding

A. Attachments

A.1 Source code for the experiments

Attachment to the thesis contains source code to the experiments we performed in the thesis. The content of the attachment includes:

- Scripts and modules for running several experiments on color clustering, image classification and generating outfits
- Source code of the web application from Chapter 3
- Documentation and a tutorial for running the scripts

The scripts for running the experiments have also been made available on the URL `http://dressandgo2018.moniq.sk`.